Escuela
Politécnica
Superior

Realidad Virtual en la Web

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
Vladyslav Kuchmenko

Tutor/es:
Domingo Gallardo Lopez

Enero 2019

Contexto

La idea de este proyecto surgió gracias a una plataforma de educación online, donde me encontré por casualidad con un curso de la Realidad Virtual en la Web y movido por mi curiosidad quise saber más sobre cómo era posible crear experiencias a priori tan demandantes en un navegador.

Mi curiosidad por la Realidad Virtual venía de antes, de hecho tenía unas gafas de realidad virtual (llamadas Lenovo explorer[1]) y ya había experimentado con varios videojuegos en esta plataforma de Windows Mixed Reality[2].

Algunas de estas experiencias eran muy impresionantes pero otras no tanto, es una tecnología relativamente nueva y todavía le faltan juegos. Y una pega que le veía es que cada videojuego pesaba varios gigas en la memoria del disco duro y seguramente lo juege solo una vez y no me guste. Otra pega es que la plataforma de Windows Mixed Reality tiene muy pocos videojuegos por lo que hay que recurrir a Steam o a la plataforma de Oculus (donde tienes que descargar un software de terceros para que sea compatible con tu plataforma).

Una vez realizado el curso de Realidad Virtual en la Web me dí cuenta que esto era perfecto para solucionar todos aquellos problemas que había tenido con la Realidad Virtual por lo que me propuse desarrollar un videojuego para la realidad virtual totalmente operativo para unas gafas de realidad virtual de alta gama, donde el usuario con un *headset* de Windows Mixed Reality, Oculus Rift, Htc Vive u otros pueda disfrutar de una experiencia de realidad virtual nada envidiable a un videojuego *stand-alone* que pudiera descargar de alguna de las tiendas de estas plataformas.

Nuestro videojuego va ha ir orientado a plataformas con *traking* de manos, sin embargo en el resto de plataformas que no dispongan de estos controles de manos seguirán pudiendo disfrutar de nuestra experiencia a través del navegador, ya sea moviéndose por el mapa con el teclado o pudiendo ver como es el juego en 3D con las cardboard.

La idea principal por lo tanto es crear un videojuego que tradicionalmente solo se podría ejecutar en plataformas muy demandantes gráficamente ,PCs *gaming,* para que se pueda ejecutar en un navegador.

Y así es como se me ocurrió la idea de crear un juego parecido a Beat Saber[3] , este juego es un juego musical de ritmo donde tienes que cortar unos cubos en la dirección que se te indique y al ritmo de la música. Uno de los principales problemas que tiene el juego es su catálogo de canciones el cual es muy limitado. Pero gracias a la comunidad y software de terceros se pueden descargar canciones con sus niveles e instalarlos para jugarlas en tu PC. Pero este proceso es muy largo, tedioso y no siempre funciona bien.

[1] Lenovo explorer, sitio oficial
[2] Windows Mixed Reality, sitio oficial
[3] Beat Saber, juego de realidad virtual en PC, sitio oficial

Para resolver este problema, en mi videojuego se automatizará la descarga de canciones y niveles usando el api pública de bsaber.com[4]. Esta página nos provee de toda la información sobre las distintas canciones que tienen ellos almacenados, también nos proveen de la canción y de un JSON con los distintos niveles jugables de la misma canción.

El JSON contiene la información sobre la temporización de las apariciones de los cubos con los que debe interactuar el jugador.

También consumiremos el api pública de bsaber.com para obtener la lista de canciones más descargadas con los distintos niveles de dificultad que tiene dicha canción.

Nuestro juego presentará la lista de canciones y una vez que el jugador haya seleccionado la canción y el nivel de dificultad que quiera jugar se descargará la canción con sus niveles de dificultad, se reproducirá la canción y se analizará el JSON, generandose los cubos para que el jugador pueda cortarlos.

[4] bsaber.com, página web que provee el API, sitio oficial

Agradecimientos

En primer lugar me gustaría dar las gracias a todos aquellos profesores que me han inspirado para seguir adelante y no tirar la toalla, en especial Domingo Gallardo López, Otto Colomina Pardo, Francisco Moreno Seco, David Tomás Díaz y Carlos Pérez Sancho. Gracias a vosotros el Grado de Ingeniería Informática es mucho más enriquecedor.

También como no a mi madre, que a pesar de no contar con muchos recursos me ha podido sacar adelante, todo lo que tengo y todo lo que soy en gran parte es gracias a ella.

Y como no gracias a mis compañeros que me han ayudado y compartido este duro camino del paso por la universidad.

Especial agradecimiento a mi pareja que me ha apoyado y ayudado en todo el proceso de la creación de este proyecto.

Índice

Índice de ilustraciones

Introducción

Para empezar vamos a ver los distintos *headsets* que existen en este momento en el mercado. Entre de los alta gama nos podemos encontrar los Oculus Rift[5], Htc vive[6], Windows Mixed Reality[7] todos ellos requieren un ordenador *gaming,* un ordenador muy potente que pueda soportar gráficos de tan alta demanda, por ejemplo un ordenador así podría costar unos 1000€, más el *headset* unos 500€ por lo que es una opción muy cara de utilizar.

Luego tenemos PlayStation VR[8] que es un accesorio de PlayStation 4[9] y que costaría unos 250€ + una PlayStation 4. También tenemos los Gear VR[10] y el Daydream[11] que requieren de un teléfono móvil compatible para poder disfrutar de las experiencias de realidad virtual, son algo más económicos unos 100€ más el teléfono móvil. Por otro lado lo que tenemos es el Google Cardboard[12], Google vende este modelo por unos 10€ pero incluso lo podemos hacer nosotros mismos en nuestra casa y lo único que necesitamos es un teléfono móvil con giroscopio para que se detecten lo giros de la cabeza.

Con esto ya nos podemos hacer una idea de las diferencias que hay entre estas distintas plataformas y *headsets,* y es que cada plataforma tiene su propia tienda donde distribuyen sus distintas experiencias por lo que si un usuario quiere disfrutar de experiencias que hay en Oculus y en HTC Vive tiene que descargar mucho software de terceros para poder hacerlo compatible, este software ocupa mucho espacio en el disco duro y baja el rendimiento de tu PC.

¿Y qué hacemos si queremos que todo el mundo pueda acceder a nuestra experiencia VR?

Tenemos un problema de distribución (por las distintas plataformas) y masividad (porque dichas experiencias ocupan mucho espacio en el disco duro). Uno de los mejores canales de distribución que tenemos hoy en día es la Web, de este modo, descubrimos el estándar WebVR[13], el cual dice a los Navegadores como Chrome o Firefox cuales son las APIs que tiene que exponer para que puedan tener una experiencia de realidad virtual en la web.

WebVR es un estándar abierto que implementan la mayoría de navegadores, por lo que podremos disfrutar en todas las plataformas anteriormente mencionadas.

Una cosa interesante es que la mayoría de experiencias de realidad virtual en la web pesan menos de 2MB (soluciona el problema de la masividad) y podemos acceder a ellas mediante un link (soluciona el problema de la distribución).

---

[5] Oculus Rift, gafas realidad virtual, sitio oficial
[6] HTC Vive, gafas realidad virtual, sitio oficial
[7] Windows Mixed Reality, gafas realidad virtual, sitio oficial
[8] PlayStation VR, gafas realidad virtual, sitio oficial
[9] PlayStation 4, consola de videojuegos, sitio oficial
[10] Gear VR, gafas realidad virtual, sitio oficial
[11] Daydream, gafas realidad virtual, sitio oficial
[12] Google cardboard, gafas realidad virtual, sitio oficial
[13] WebVR, estándar abierto para los navegadores, sitio oficial

Otra ventaja de trabajar con VR en la web es que ahora no tenemos que inventar nuevas tecnologías. WebVR está basado en WebGL[14] que ya existe en la web y la mayoría de las experiencias de WebVR se basan no solamente en WebGL, sino que también en Three js[15], Three js es el framework de WebGL más común en la web, si ya hemos trabajado con 3D en la web no nos resultará difícil crear experiencias VR.

Y algo aún más interesante todavía es que gracias a esta facilidad surgieron distintos frameworks para poder trabajar a más alto nivel y uno de ellos es A-Frame[16], es de Mozilla y nos permite hacer VR usando solo HTML y Javascript.

Otra cosa más es que no solo nos vamos a aprovechar de A-Frame y su facilidad de crear experiencias VR sino que también se pueden incorporar otras librerías como React[17], Vue JS[18] e incluso frameworks como Angular[19] que nos permiten trabajar con HTML y JavaScript y de esta manera crear experiencias VR como si de una aplicación web se tratara.

[14] WebGL, api estándar que define el renderizado de gráficos 3D en el navegador, sitio oficial
[15] Three JS, framework que implementa WebGL, sitio oficial
[16] A-Frame, framework de alto nivel para la creación de experiencias VR, sitio oficial
[17] React, biblioteca javascript para creación de aplicaciones web, sitio oficial
[18] Vue JS, biblioteca javascript para creación de aplicaciones web, sitio oficial
[19] Angular, framework para la creación de aplicaciones web, sitio oficial

Estado del arte

En los apartados anteriores vimos un poco porque tiene sentido hacer VR en la web y ahora voy a justificar eso mostrando cuáles son algunas de las mejores experiencias que he visto de web vr.

A-Painter

Vamos a empezar por Tilt Brush[20] de Google que nos permite hacer arte, pintar y dibujar en nuestro propio ambiente.

El equipo de Mozilla VR[21] que es el que hizo A-Frame, el framework que vamos a utilizar en el proyecto, hizo A-Painter[22] este videojuego justamente lo que hace es replicar la idea de Tilt Brush pero en el navegador. Nosotros entramos al sitio web de A-Painter y con unas gafas de realidad virtual como HTC Vive podemos empezar a pintar.

Hasta este punto es todo muy parecido, pero que pasa si queremos compartir nuestra creación con alguien ? En el caso de Tilt Brush te genera un archivo el cual mandamos a la persona que queremos que vea nuestro dibujo, pero tenemos ciertas limitaciones y es que la persona que quiera ver el dibujo también tendrá que tener unas HTC Vive y el propio juego para poder ver dicho dibujo.

En el caso de A-Painter lo que compartiremos es un link y la persona que quiera ver nuestro dibujo, y no necesita más que un ordenador o un teléfono móvil para poder disfrutar de nuestra creación.

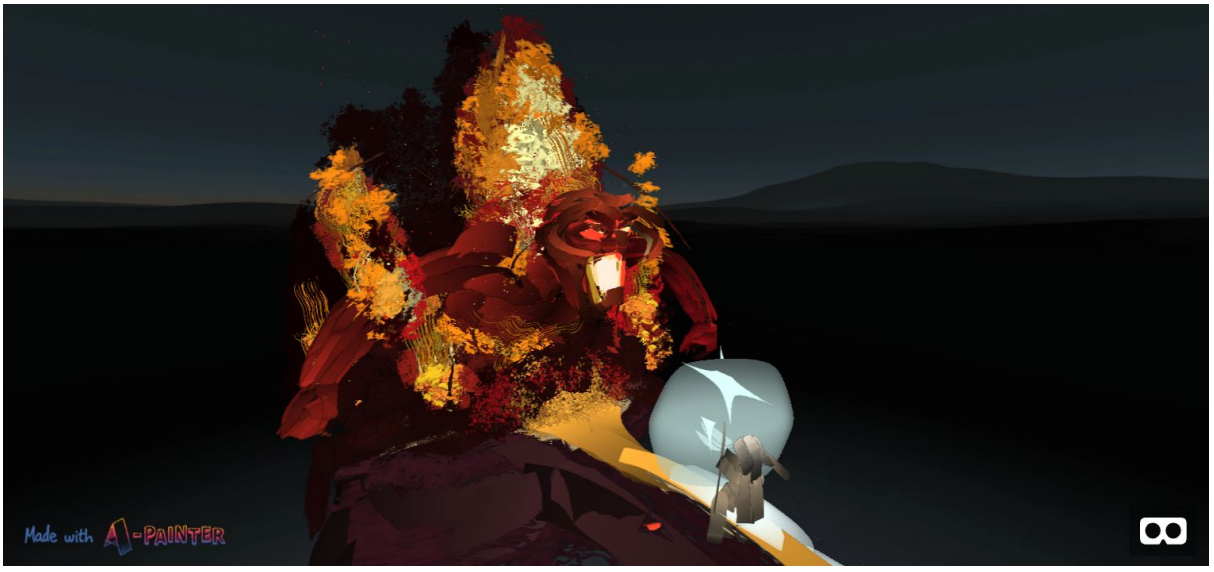


Ilustración 1 - A-Painter

[20] Tilt Brush, juego de realidad virtual *stand-alone* , sitio oficial
[21] Mozilla VR, equipo de desarrollo de realidad virtual en la web, sitio oficial
[22] A-Painter, el juego equivalente del Tilt Brush, sitio oficial

A-Blast

Otro juego muy interesante es A-Blast[23], y es muy interesante porque es un juego muy inmersivo e interactivo, se trata de destruir unos muñequitos que te disparan con las dos pistolas que tienes en las manos. Y este juego refleja a la perfección de la ventajas que tiene web vr y es que solo tienes que entrar a un link y ponerte a jugar una experiencia muy chula mientras que si la quisiéramos probar en alguna plataforma de realidad virtual tendriamos que descargarnos todo el juego, instalarlo, y si no nos gusta y esto pasa mucho, desinstalarlo.

Ilustración 2 - A-Blast

aframe.io

Estos dos últimos juegos son producto del equipo de Mozilla VR y como no, tienen muchas más experiencias las cuales las puedes encontrar en la página oficial de A-Frame, aframe.io [24]

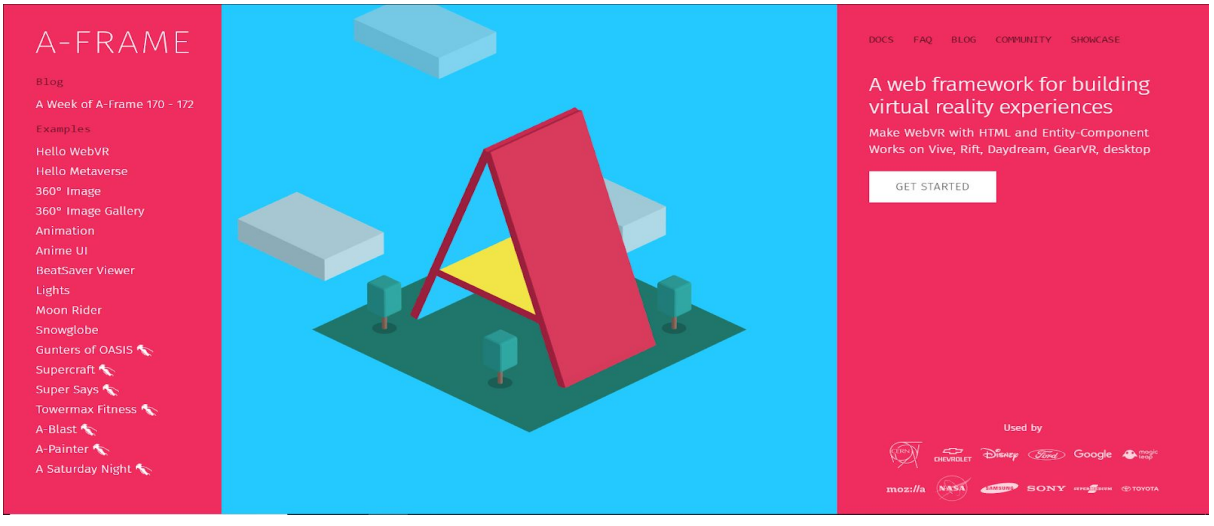


Ilustración 3 - A-Frame

[23] A-Blast, juego realidad virtual en la web, sitio oficial
[24] aframe.io, sitio web donde se encuentra toda la documentación y las experiencias oficiales de mozilla vr, sitio oficial

Google web vr experiment

Y por último cabe mencionar que Google también está realizando experiencias en web vr , un ejemplo de ello Web VR experiments[25] es una página web que recopila distintas experiencias en Web VR como por ejemplo un juego a tenis de mesa, un paseo por Marte, fuegos artificiales o música en 8D[26].

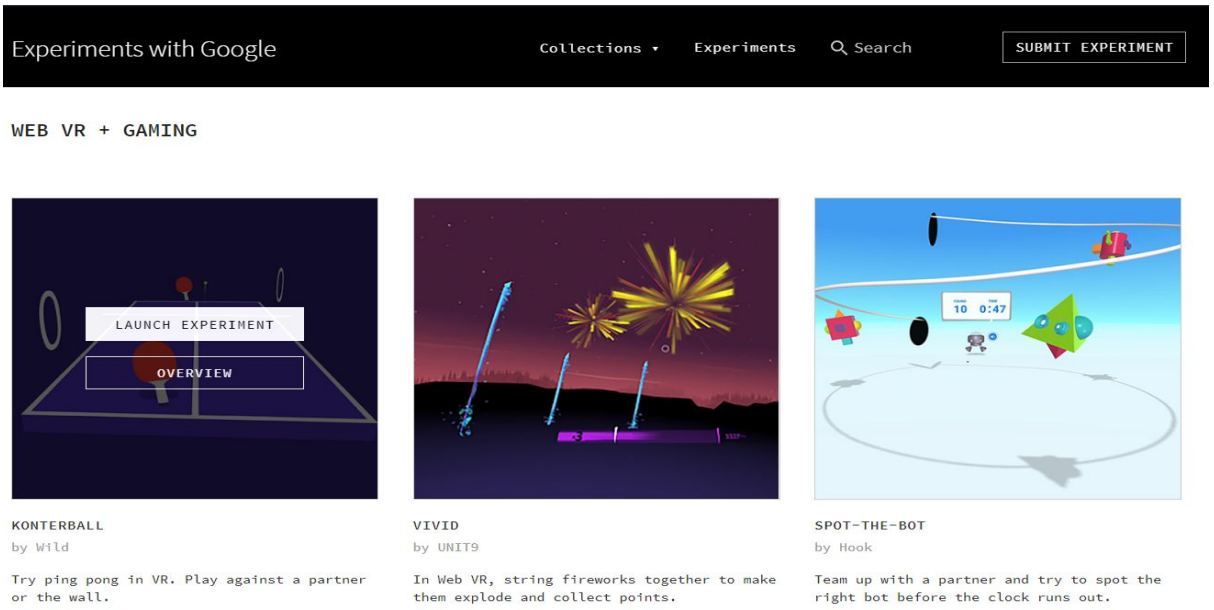


Ilustración 4 - Experiments with Google

[25] Web vr experiments, página web que recopila distintas experiencias en web vr, sitio oficial
[26] Música en 8D, escuchar música como si estuvieras en el estudio de grabación, explicación que es la música en 8D

Objetivos

En este punto y habiendo visto cual es el contexto de la realidad virtual a dia de hoy, cuales son su principales problemas y cuál podría ser la solución a esos problemas, así como las distintas experiencias que nos ofrece el web vr, me dispongo a establecer unos objetivos claros en los que se basará el proyecto a partir de ahora.

Integración de frameworks

Conseguir que la forma de trabajar con el framework de Angular 8 encaje fluidamente y sin provocar ningún problema en el desarrollo del proyecto con la forma de trabajar con el framework de Aframe, ya que estos dos frameworks no se suelen utilizar juntos y no hay mucha información, por lo que habrá que investigar e implementar dicha forma de trabajar.

Idea general y mecánicas del videojuego

Una vez que tenemos claro qué clase de videojuego queremos implementar hay que establecer qué elementos va a contener, cuales son los escenarios que lo van a formar y cuáles son las mecánicas del videojuego que hacen que jugarlo sea divertido y entretenido.

Gracias a unas buenas mecánicas aunque el juego pueda parecer simple gráficamente puede ser aún mejor que uno con mejores gráficos.

Un claro ejemplo de ello es él muy conocido por todos Minecraft[27] que aún teniendo unos gráficos muy simples triunfa por sus increíbles mecánicas de juego, lo que nos hace ver la importancia de diseñarlas e implementarlas.

Creación de tareas

Cuando ya tenemos claro cuales son los elementos, escenarios y mecánicas que se quieren implementar vamos a transformar toda esta información en tareas que realizar para llevar a cabo el proyecto, estas tareas estarán expresadas en forma de Historias de Usuario[28].

Una vez creadas las tareas se definirá un MVP(producto mínimo viable) que sería la implementación mínima necesaria para poder presentarla como la primera versión del videojuego.

[27] Minecraft, videojuego sitio oficial
[28] Historias de Usuario, forma ágil de definir características o (*features*), explicación más detallada.

Implementación de las tareas

Nuestro objetivo en esta parte es utilizando las tecnologías de Angular 8, Aframe, Typescript, Javascript y HTML crear un videojuego para el mundo de la Realidad Virtual en la Web y de esta forma contribuir en aumentar el catálogo de experiencias en esta nueva plataforma.

La implementación de estas tareas se llevará a cabo siguiendo algunas de los valores, principio y prácticas de las metodologías Ágiles[29], más concretamente el TDD[30] y Scrum[31].

Conclusión

En conclusión lo que me propongo es llevar a cabo un proyecto de la creación de un videojuego desde su concepción hasta su implementación utilizando unas tecnologías novedosas y unas metodologías de gestión de proyectos igual de innovadoras .

[29] Metodologías Ágiles, conjunto de valores, principio y prácticas, explicación más extensa
[30] TDD, Test Driven Development, explicación más extensa
[31] Scrum, framework ligero, empírico e iterativo, explicación más extensa

Tecnologías

En este apartado vamos a hablar sobre las principales tecnologías gracias a las cuales es posible realizar nuestro proyecto, las cuales son A-Frame y Angular 8.

A-Frame

A-Frame es un framework para la creación de experiencias de realidad virtual donde los desarrolladores pueden crear escenas 3D y de WebVR usando HTML.

El HTML proporciona una herramienta familiar para los desarrolladores web, pero además al incorporar el patrón arquitectónico (Entidad Componente Sistema)[32], utilizado por motores como Unity, se convierte en una herramienta poderosa a la vez que fácil de utilizar.

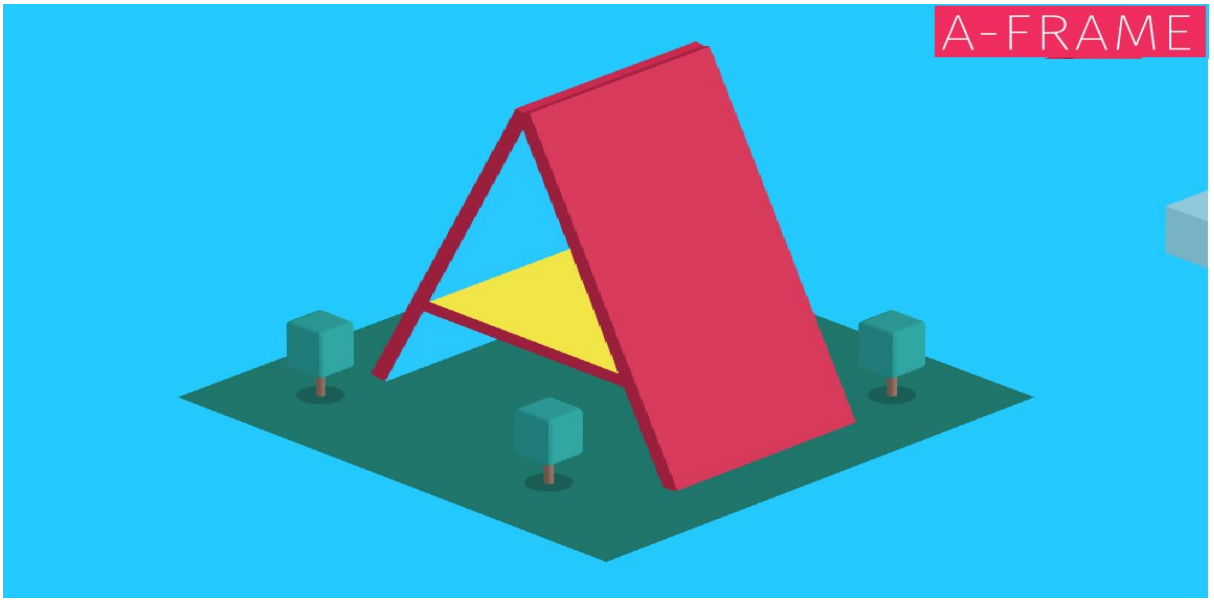


Ilustración 5, Logo A-Frame

Este framework soporta la mayoría de visores de realidad virtual tales como HTC Vive, Oculus Rift, Oculus Go, Windows Mixed Reality, Daydream, Gear VR, Cardboard etc... E incluso se puede utilizar para experiencias de Realidad Aumentada[33].

El objetivo del framework es realizar experiencias de realidad virtual totalmente inmersivas que van desde algunas tan simples como imágenes o videos en 360º hasta otras tan complejas que hacen uso del posicionamiento en las tres dimensiones del visor con sus controladores de manos.

[32] Entidad Componente Sistema, patrón arquitectónico, explicación más detallada
[33] Realidad Aumentada, mezcla entre realidad virtual y la realidad, explicación más detallada

Algunos de los beneficios de utilizar dicho framework son los siguientes:

- La posibilidad de crear contenido 3D y VR solo utilizando HTML sin necesidad de tener ningún paso previo de construcción.
- El poder configurar la escena mediante una línea de HTML esto incluye el renderizado del lienzo, render loop, luces, controles y configuración del WebVR.
- La posibilidad de invocar una herramienta de inspección visual desde cualquier escena.
- La compatibilidad con la mayoría de bibliotecas y frameworks para la creación de interfaces de usuario en la web, tales como React, Angular, Vue.js.
- Su muy extensa y bien organizada documentación donde podemos encontrar absolutamente todo lo que podemos hacer con dicha herramienta.
- Su magnífica comunidad, donde entrando a su canal de Slack[34] y dejando tu pregunta alguno de los desarrolladores o bien alguno de los entusiastas del framework responderán tu duda.
- Compatibilidad con todo tipo de visores y sus respectivos controladores de manos si los tienen y sino control con la mirada.
- El rendimiento, gracias a una serie de optimizaciones somos capaces de obtener hasta 90 fps[35] en un navegador.

Y para finalizar vamos a ver un ejemplo de cómo podemos crear una escena de realidad virtual con A-Frame.

Hello.html

```
<html>
 <head>
  <script src="https://aframe.io/releases/0.9.2/aframe.min.js"></script>
 </head>
 <body>
   <a-scene>
     <a-box position="-1 0.5 -3" rotation="0 45 0"color="#4CC3D9">
     </a-box>
     <a-sphere position="0 1.25 -5" radius="1.25" color="#EF2D5E">
     </a-sphere>
     <a-cylinder position="1 0.75 -3" radius="0.5" height="1.5"
color="#FFC65D">
     </a-cylinder>
     <a-plane position="0 0 -4" rotation="-90 0 0" width="4" height="4"
color="#7BC8A4">
     </a-plane>
     <a-sky color="#ECECEC"></a-sky>
   </a-scene>
 </body>
</html>
```

[34] Slack, cliente de mensajería instantánea, sitio oficial
[35] fps, fotogramas por segundo, explicación más detallada

Y lo que obtenemos es la siguiente escena si abrimos el archivo html en nuestro navegador

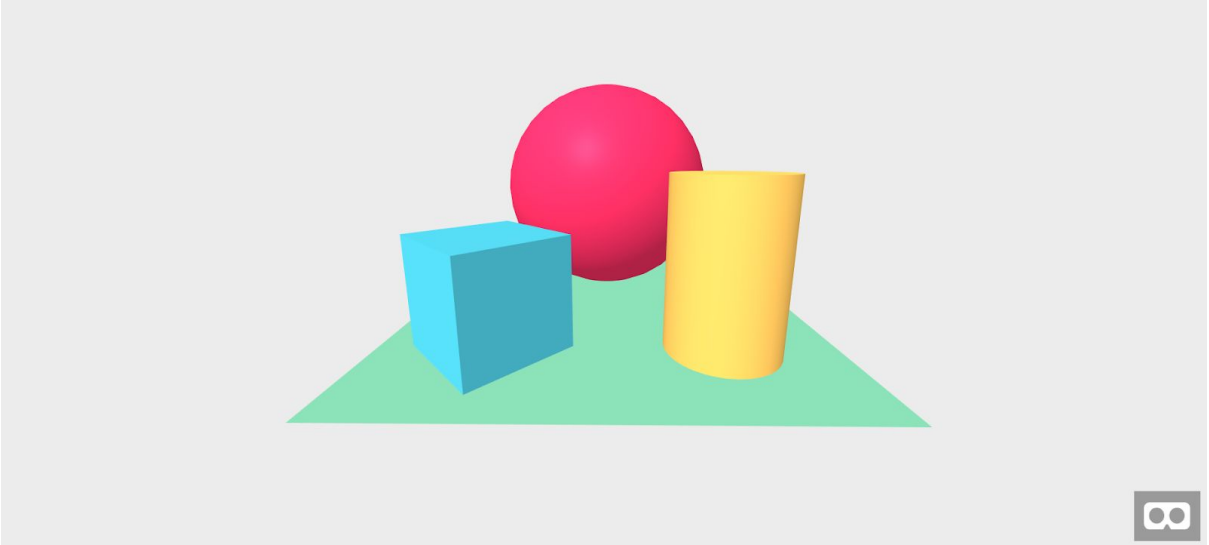


Ilustración 6 - Ejemplo experiencia WebVR con A-Frame

Como podemos ver obtenemos una esfera, un cilindro, un cubo y un plano, todo ello simplemente con 4 etiquetas html (a-sphere, a-cylinder, a-box, y a-plane) y las propiedad de posición (position), rotación (rotation) y color (color).

Si nos fijamos abajo a la derecha tenemos un icono de unas gafas de realidad virtual, si lo pulsamos dependiendo del dispositivo que estemos utilizando entraremos al modo de realidad virtual. Si estamos en el ordenador pasará a pantalla completa, si estamos en el móvil la pantalla se dividirá en dos para ponerlo en unas cardboard y si estamos con Oculus o Vive entraremos dentro de la escena incluso pudiendo movernos adelante y atrás dentro de la propia escena.

Esto por supuesto no solo queda aquí, podemos añadirle controladores de manos si el dispositivo los tiene, podemos crear nuestros propios componentes más complejos que una esfera o un cubo, podemos utilizar componentes de otras personas que encontremos en la web, podemos añadirle físicas y detección de colisiones entre objetos y muchas más cosas.

Angular 8

Angular es un framework para el desarrollo de aplicaciones web, está escrito en Typescript y es mantenido por Google. Se maneja un desarrollo basado en modelo vista controlador (MVC) y la ejecución se lleva del lado de cliente (navegador).

Angular 8 es la octava versión de Angular que es a su vez la evolución hecha totalmente desde cero de AngularJS.

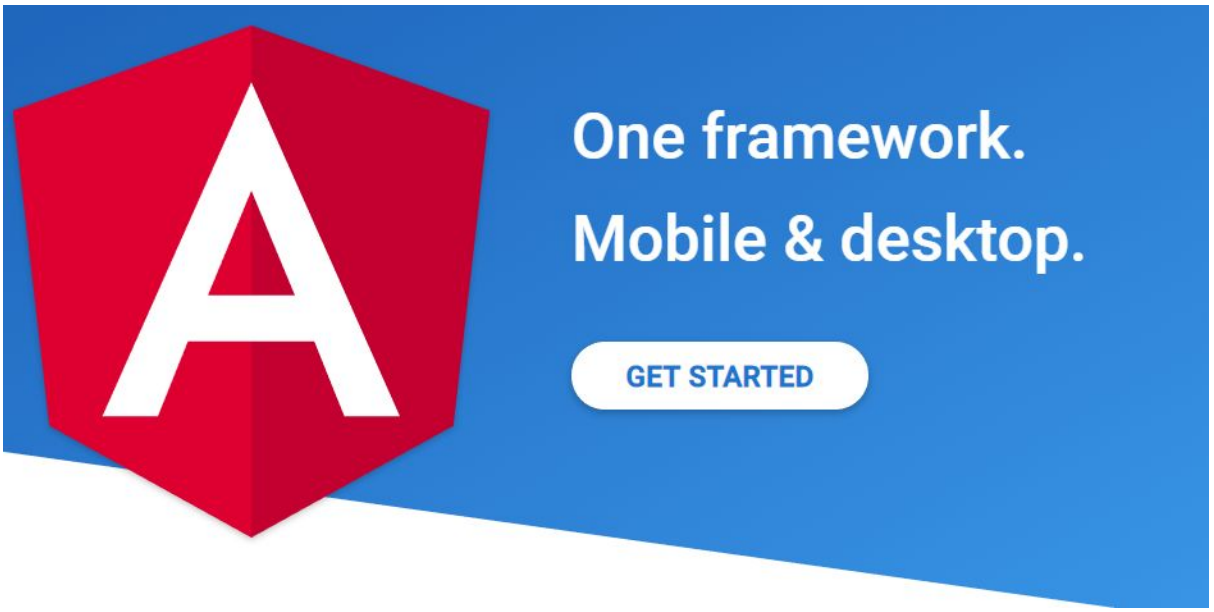


Ilustración 7 - Logo Angular

Usa Typescript como lenguaje de programación. Typescript es un subconjunto de Javascript que esencialmente añade tipos estáticos y objetos basados en clases.

Se pueden crear interfaces interactivas con muy poco esfuerzo, ya que el framework se encarga de actualizar las vistas, los componentes y los servicios, todo ello de manera muy eficiente.

La filosofía del framework es la creación de aplicaciones de una sola página, esto quiere decir que la navegación entre distintas secciones de la página así como la carga de datos se realizan de manera dinámica lo que permite hacerlo sin refrescar la página en ningún momento.

A continuación podemos ver los 8 mayores bloques de la arquitectura de dicho framework:

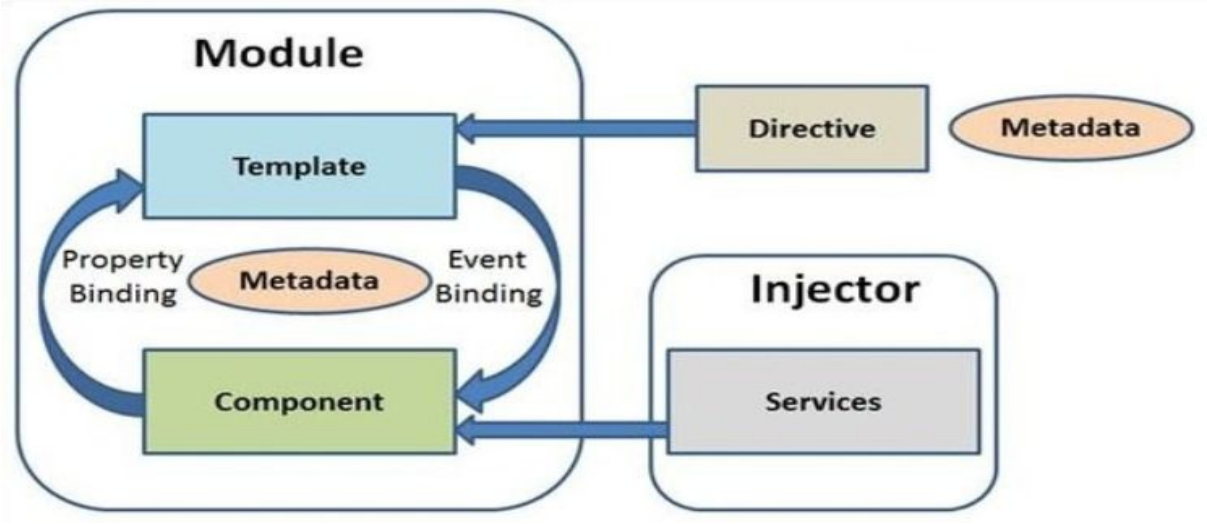


Ilustración 8 - Arquitectura de Angular Sitio del que se ha extraído la imagen

El módulo es muy parecido a una clase. Se puede describir como un bloque de código el cual se utiliza para desarrollar una tarea específica.

El componente es la vista de la aplicación y la lógica de la página.

La plantilla es la parte principal, esta nos proporciona el aspecto del componente, es decir, la vista del componente se define usando la plantilla.

Los metadatos se utilizan para ampliar la funcionalidad de la clase,por ejemplo, para definir cualquier componente de Angular se utilizan metadatos de la clase (@Component decorator).

El *data binding* es una de las caracteristicas más poderosas del framework, siendo la conexión entre el modelo y la vista. Consigue la sincronización automática de datos.

Las directivas son atributos HTML personalizados que se utilizan para extender la funcionalidad de un componente.

Los servicios se utilizan para compartir datos y funcionalidad entre distintos módulos. Comúnmente son Singleton aunque se puede forzar a que sean Multiton, y son inyectados al componente por el framework.

La inyección de dependencias se utiliza para adjuntar funcionalidad a los componentes en tiempo de ejecución. Los objetos se pasan como dependencias. Gracias a la inyección de dependencias los componentes se pueden manipular, reutilizar y testear fácilmente.

Conclusión

En conclusión el A-Frame framework nos va a facilitar la tarea de creación de nuestro videojuego y el framework de Angular lo que va a proporcionarnos es que el código que crearemos estará mucho más estructurado, hará que sea reutilizable y mantenible a largo plazo además de estar muy bien optimizado.

Metodología

Dadas las tendencias actuales en el mundo de la gestión de proyectos informáticos hemos decidido optar por una metodología ágil que combina caracteristicas de Scrum[36] y Kanban[37].

Scrum y Kanban

En este proyecto se ha hecho una combinación entre el framework Scrum y el método Kanban.

Algunas de las prácticas que hemos realizado han sido las siguientes:

- Se han realizado una serie de sprints, reuniones cada 2 o 3 semanas con el profesor.
- En dichas reuniones se hacía una retrospectiva del trabajo realizado y el trabajo pendiente que pudiera quedar.
- Se realizaba una pequeña demostración, algo que funcionara y pudiera ser visible de cara al público.
- Se planificaban las tareas para el siguiente sprint.
- Se creó un product backlog.
- Se obtuvieron gráficas *burndown*.

De Kanban tomamos su tablero y sus métricas del trabajo realizado y el trabajo pendiente por realizar. No hacía falta limitar el trabajo en progreso ya que el componente del equipo solo es una persona y como se ha de enfocar solo en una tarea a la vez el trabajo en progreso en una columna siempre era uno.

Wiki[38]

Ilustración 9 - Apartado Wiki de Github

Wiki es una sección del repositorio de GitHub donde hemos dejado toda nuestra documentación funcional que hemos elaborado sobre el proyecto. Esto lo que nos permite es que si en algún momento del desarrollo tenemos alguna duda sobre cómo ha de funcionar aquello que estamos haciendo podemos consultarlo de una forma fácil.

[36] Scrum, framework ligero, empírico e iterativo, explicación más extensa
[37] Kanban, es un método para mejorar los procesos, explicación más extensa
[38] Wiki, sección del Github que te permite dejar documentación sobre el proyecto, sitio oficial

Épicas de usuario

En primer lugar lo que vemos nada más entrar en la Wiki es una lista con las Épicas de Usuario y su descripción.

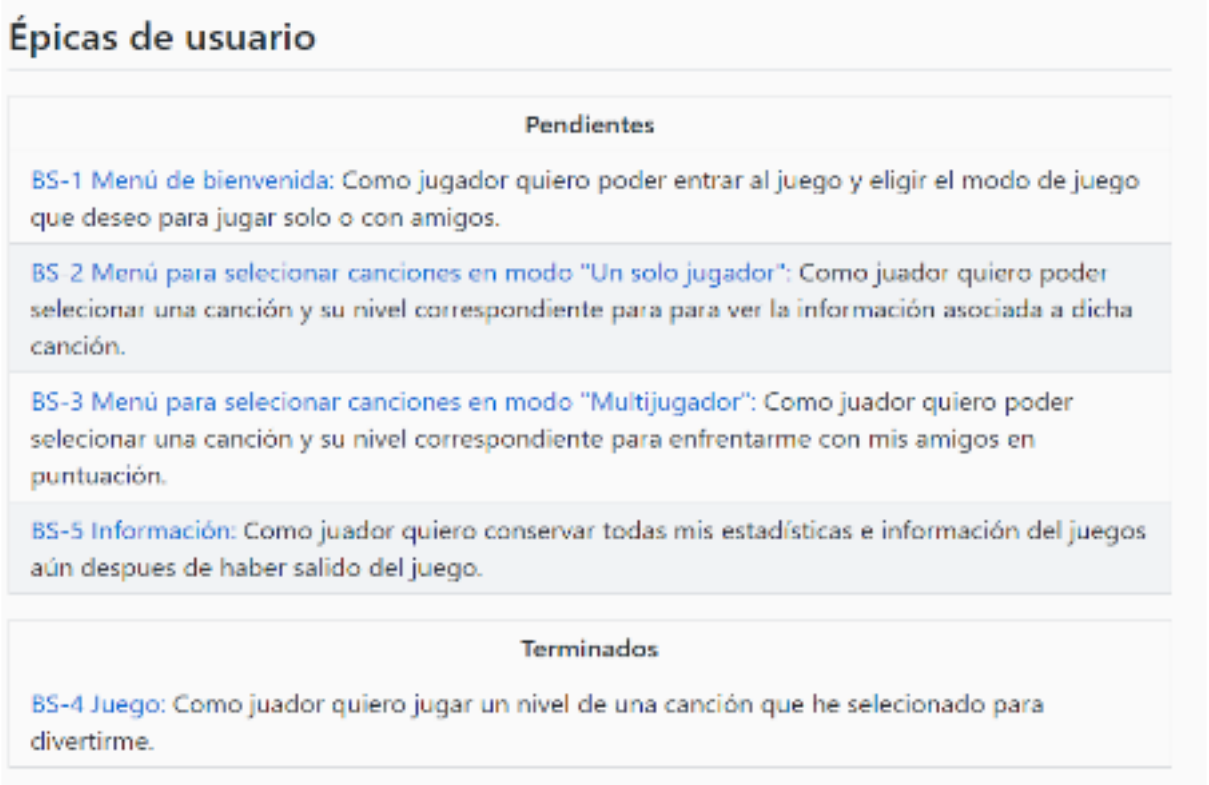
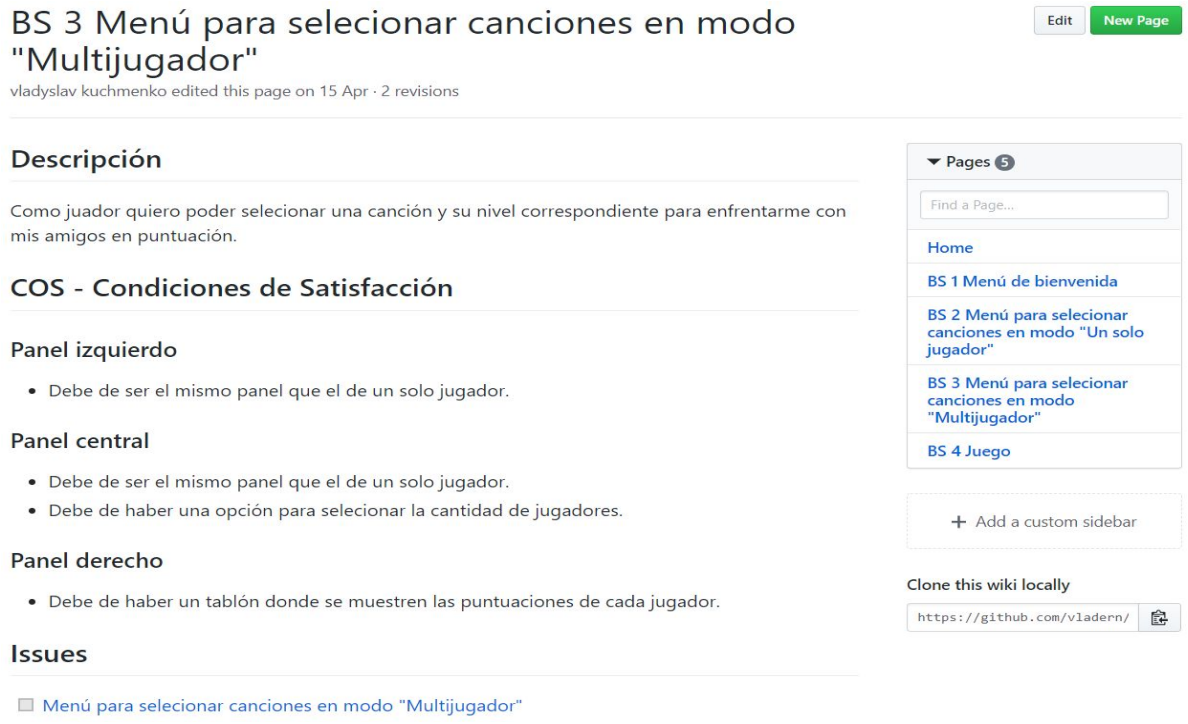


Ilustración 10 - Épicas de usuario

Tal y como podemos observar en la imagen hay dos listas de épicas de usuario, la primera con las épicas que todavía no están terminadas y otra con la que ya están terminadas.

Otra cosa interesante es que si le damos click en el enlace de una de las épicas de usuario nos lleva al detalle de la misma.



La descripción de cada una de las épicas sigue el siguiente patrón:

<Título>
Como <tipo de usuario>
quiero <acción>
para <resultado, objetivo>

Esta forma de describir las épicas de usuario es importante ya que nos centramos en la funcionalidad que será valiosa para el usuario final.

Dado que las historias de usuario han de ser testables el siguiente apartado de la descripción son las condiciones de satisfacción donde enumeramos cuales son las condiciones que ha de cumplir dicha historia de usuario para poder darse por terminada.

Y por último tenemos una lista de las historias de usuario que conforman la épica de usuario, si le damos click nos lleva a dicha historia de usuario en particular.

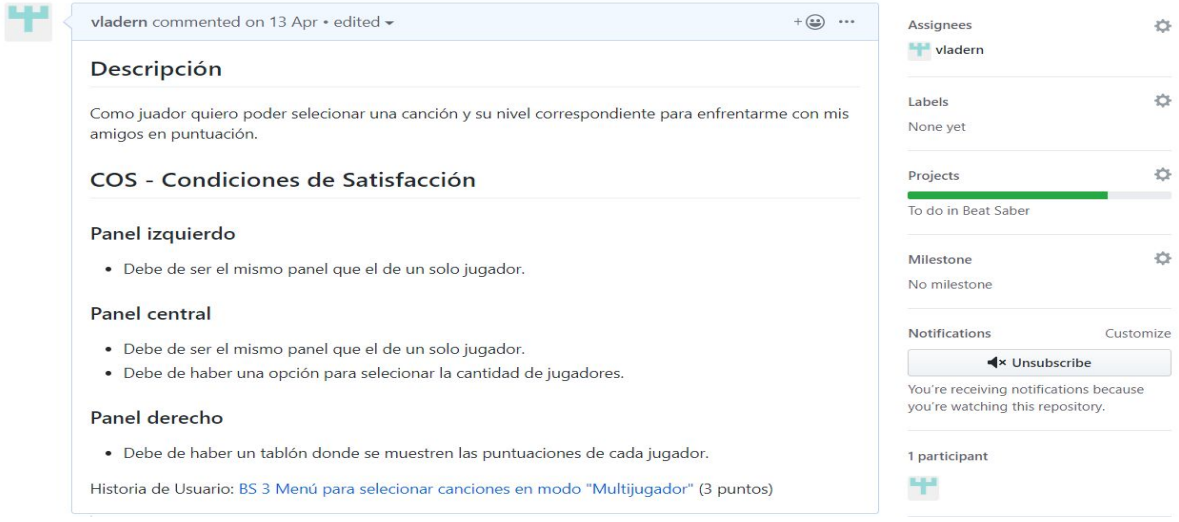


Ilustración 11- GitHub issue

Como podemos ver la estructura de dicha historia de usuario es exactamente la misma que la de una épica de usuario, tenemos una descripción, las condiciones de satisfacción y un enlace que nos lleva a la épica de usuario a la que pertenece. Pero además podemos ver que se le ha asignado una puntuación que corresponde a la dificultad de esta historia de usuario.

Estimación del tamaño de historias de usuario

No es otra cosa que una lista con todas las historias de usuario del proyecto con los puntos de dificultad que le corresponden, estas puntuaciones fueron estimadas por el equipo de desarrollo y siguen la sucesión de fibonacci.

Estimación del tamaño de historias de usuario	
Issue	Points
Panel izquierdo del menu inicial	1
Panel central del menu inicial	2
Panel derecho del menu inicial	3
Panel izquierdo del menu de seleción de la canción	3
Panel central del menu de seleción de la canción	5
Panel derecho del menu de seleción de la canción	2
Menú para selecionar canciones en modo "Multijugador"	3
El jugador debe permanecer en el centro de la plataforma de juego	2
Al comenzar el juego debe de sonar la musica que ha selecionado el jugador	3
El jugador debe interactuar con los cubos	5
El jugador debe esquivar obstaculos	3
Recibir puntos	2
Fallas el nivel	2
Cortar la racha	1
Total	**47**

Ilustración 12 - Estimación del tamaño de historias de usuario

Producto mínimo viable

Al igual que la anterior es una lista formada por historias de usuario, pero en esta lo que se refleja los las funcionalidades mínimas necesarias para poder sacar a la luz el proyecto.

Producto mínimo viable	
Panel central del menu inicial	2
Panel central del menu de seleción de la canción	5
El jugador debe permanecer en el centro de la plataforma de juego	2
El jugador debe de tener un sable en cada mano	2
Al comenzar el juego debe de sonar la musica que ha selecionado el jugador	3
El jugador debe interactuar con los cubos	5
El jugador debe esquivar obstaculos	3
Recibir puntos	2
Fallas el nivel	2
Cortar la racha	1
Total	**27**

Ilustración 13 - Producto mínimo viable

Tablero kanban

El objetivo principal del tablero es mostrar visualmente en que está trabajando el equipo en cada momento.

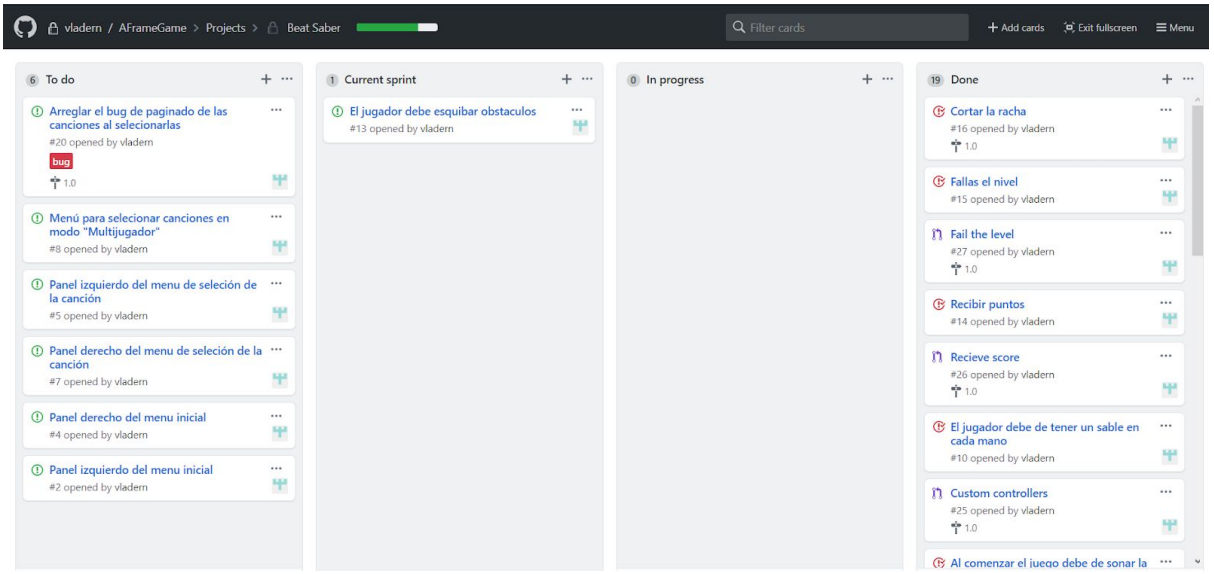


Ilustración 14 - Github tablero kanban

Este es el tablero que hemos utilizado en nuestro proyecto, se puede observar una columna que hace de Backlog (lista con todas las tareas del proyecto) es la primera y se llama ***To do*****,** la siguiente columna se llama ***Current sprint*** y como su nombre indica en esa columna se colocan las tareas que vamos a realizar en el sprint actual, a continuación está la columna ***In progress*** donde movemos las tareas que estamos realizando actualmente y por último tenemos las columna ***Done*** donde se colocan las tareas que ya han sido finalizadas.

Flujo de trabajo

El flujo de trabajo en este proyecto es muy sencillo ya que el equipo está formado solo por un integrante. Por lo que hemos escogido una sola rama principal ***master*** donde tenemos las últimas caracteristicas de nuestro proyecto y para cada una de las historias de usuario que vayamos a implementar creamos una nueva rama con un nombre descriptivo de la misma en esta rama hacemos nuestro desarrollo de la historia de usuario y ya que utilizamos TDD[39] no es necesario el testeo posterior de nuestro código, por lo que cuando se da por finalizada la nueva característica del código se hace un ***pull request*** a la rama principal. Y cuando se aprueba se mezcla nuestra rama con la rama principal.

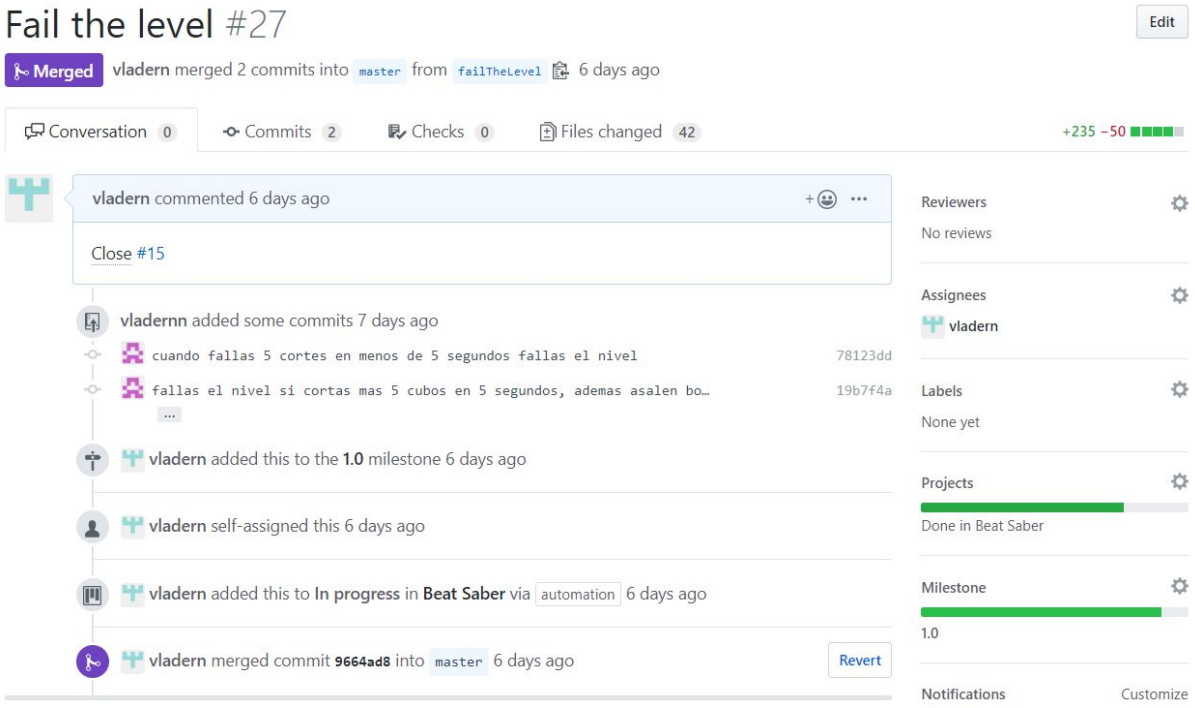


Ilustración 15 - Github pull request

En esta imagen podemos ver un ejemplo de un ***Pull Request***[40] **,** por un lado dejamos un comentario Close #Num_of_issue de esta forma GitHub cierra la historia de usuario automáticamente cuando se hace el merge en la rama principal pero además mueve dicha historia de usuario a la columna de ***Done.*** A continuación vemos una imagen con lo que explicamos anteriormente del flujo de las ramas de git pero de forma visual.

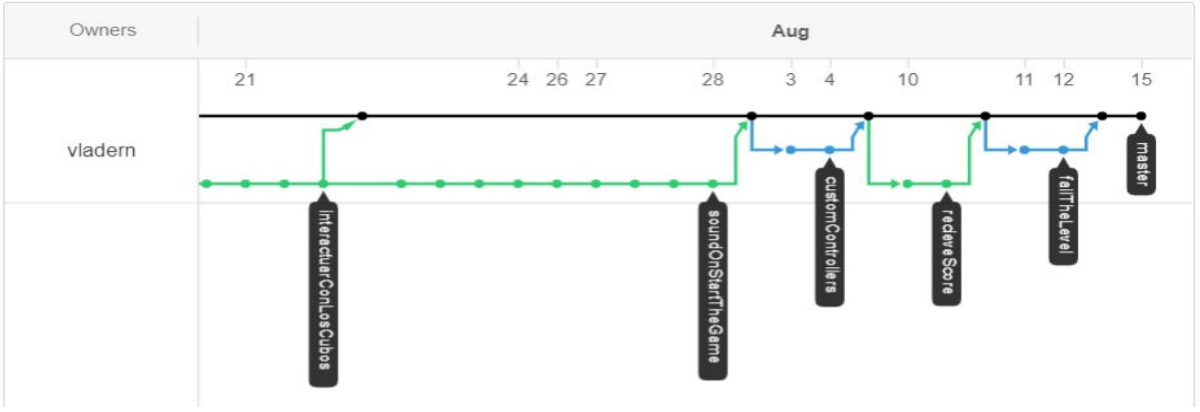


Ilustración 16 - Flujo de trabajo git representado visualmente

[39] TDD, Test Driven Development, explicación más extensa
[40] Pull Request, petición para un merge, sitio oficial

Documentación

Al principio de los 2000 estaba de moda documentar absolutamente todo.Bien, mediante, enciclopedias la cuales nadie leía y mucho menos se mantenía. En los últimos años se ha puesto de moda, con la excusa de que somos ágiles, de no documentar absolutamente nada. Y como todo, ninguna de ambas opciones es correcta.

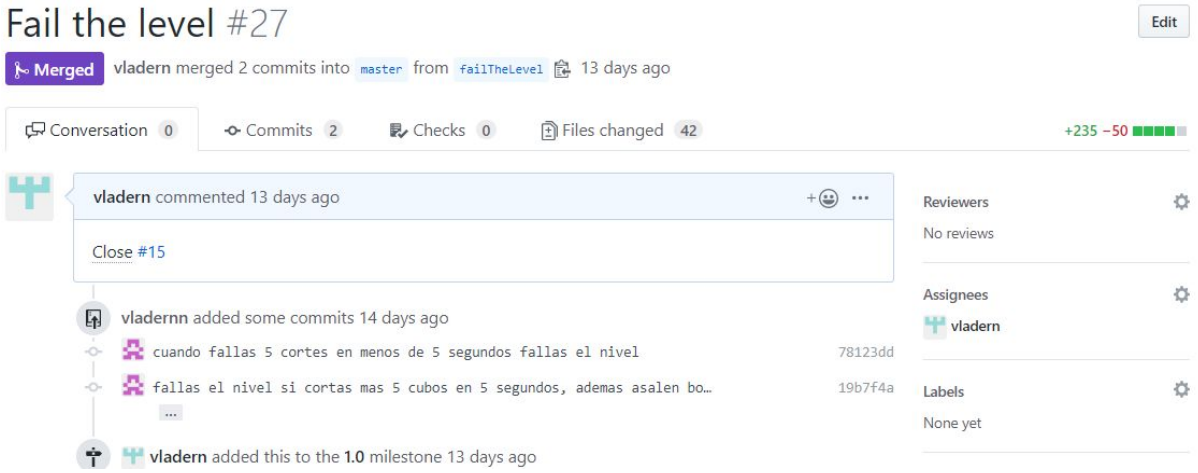
La razón por la que no es necesaria una documentación voluminosa es la dificultad de mantener y sincronizarla con algo tan dinámico como el software, el tiempo que se utiliza para documentar se puede invertir en crear nuevas caracteristicas (“software que funcione, por encima de documentación exhaustiva”). Sin embargo la preferencia del manifiesto ágil por el código, sobreponiéndose a la documentación solo aplica a la documentación de diseño: documentos voluminosos y detallados que describen implementaciones previas a comprender los problemas que se tienen que solucionar.

Se necesita documentación a mayor nivel de abstracción, sin tantos detalles tecnológicos innecesarios y tan relacionados al dominio y los requisitos. Estos documentos nos ayudarán a ver mejor el bosque que se oculta en el mar de código. Además esta documentación ha de poder modificarse y mantenerse de una manera fácil y rápida.

Por estas razones elegimos la Wiki de Github como el lugar de documentar nuestro proyecto, sobre qué es la Wiki ya lo vimos en el apartado de Metodología.

En la Wiki podemos encontrar toda la documentación funcional del proyecto. Esta tiene cierta jerarquía, primero están las épicas de usuario, donde describimos la funcionalidad de una parte muy grande del sistema, estas épicas de usuario están compuestas por historias de usuario, esto ya describe la funcionalidad de un trozo de la épica de usuario, esta historia de usuario es la que será desarrollada.

Cuando desarrollamos una de estas funcionalidades lo hacemos en una rama de Git y cuando la terminamos la integramos en la rama principal, esta integración se llama en GitHub un *Pull Request* en este pull request podemos dejar un comentario y en dicho comentario escribimos que historía de usuario es la que hemos desarrollado.



De esta forma tenemos relacionada nuestra funcionalidad con el código que se ha generado para la implementación de dicha funcionalidad y si son necesarios más detalles técnicos, como se ha implementado una funcionalidad en particular, estos son fáciles de encontrar, mantener y actualizar.

Arquitectura

Por lo que respecta a la arquitectura podemos descomponer la de este proyecto en dos capas o niveles.

AFrame

Tal y como ya se ha comentado en apartados anteriores, AFrame hace lo dificil facil, haciendo posible que usando etiquetas HTML se creen experiencias VR.

Pero obviamente esto no podía ser tan fácil, AFrame es un framework Componente-Entidad-Sistema donde una definición básica de ello podría ser:

Las **entidades** son la base de todos los objetos de la escena, funcionan como objetos contenedores a los que se les puede añadir más componentes. Sin componentes las entidades no hacen ni representan nada son como s vacíos. Se representan por el elemento y prototipo **<a-entity> .**

Los **componentes** son módulos reutilizables o contenedores de datos que se pueden conectar a entidades para proporcionar apariencia, comportamiento y/o funcionalidad. Toda la lógica se implementa a través de los componentes, de esta forma podemos definir distintos tipos de objetos mediante la mezcla o unión de distintos componentes y se representan por los atributos HTML.

Los **sistemas** proveen un ámbito global, gestion y servicios para las clases del componente, muchas veces son opcionales pero se pueden utilizar para separar la lógica y los datos, manejan la lógica y los componentes actúan como contenedores de datos. Al igual que los componentes se representan por los atributos HTML.

La sintaxis de una entidad tiene que seguir la siguiente definición:

```
<a-entity ${componentName}="${propertyName1}: ${propertyValue1};
${propertyName2}: ${propertyValue2}"></a-entity>
```

Con esto podemos añadir mas componentes para añadir apariencia, comportamiento o funcionalidad. O podemos actualizar los datos del componente para configurar una entidad.

Algunos ejemplos abstractos de entidades que se han obtenido a partir de la composición de distintos componentes son:

- `Box = Position + Geometry + Material`
- `Ball = Position + Velocity + Physics + Geometry + Material`
- `Player = Position + Camera + Input + Avatar + Identity`

En esta imagen podemos ver de forma visual este concepto de la composición.

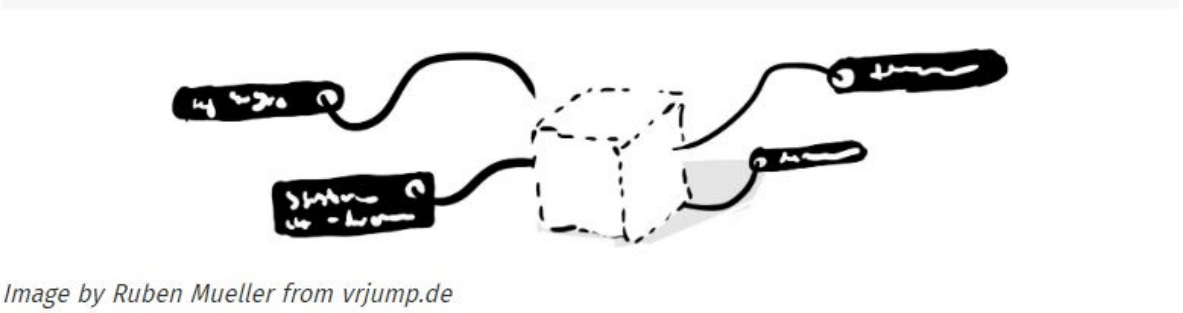


Ilustración 17 - entidad en AFrame

Esta forma de desarrollar lo que permite es que los componentes sean muy reutilizables por lo que muchos de los componentes que vamos a utilizar son de la propia comunidad.

En conclusión el framework tiene entidades, componentes y sistemas los cuales los podemos tratar como elementos HTML del DOM normales y corrientes y para ello normalmente se utiliza el lenguaje de programación JavaScript.

Angular

En el apartado de tecnologías ya hemos visto cuál es la arquitectura del framework pero hay muchas formas de trabajar con Angular y en este apartado vamos a ver como se ha hecho en este proyecto.

En primer lugar tenemos los componentes con los cuales definimos el aspecto y el comportamiento de todos los elementos que vamos a visualizar en nuestro proyecto. En el componente principal (el que engloba toda la aplicación) es el lugar donde se define la entidad de la escena (<a-scene></scene>) y aquí es donde empezamos a utilizar componentes y entidades del Aframe para construir nuestra experiencia VR en la web. Definimos también la entidad del jugador y los distintos escenarios que vamos a tener en el juego.

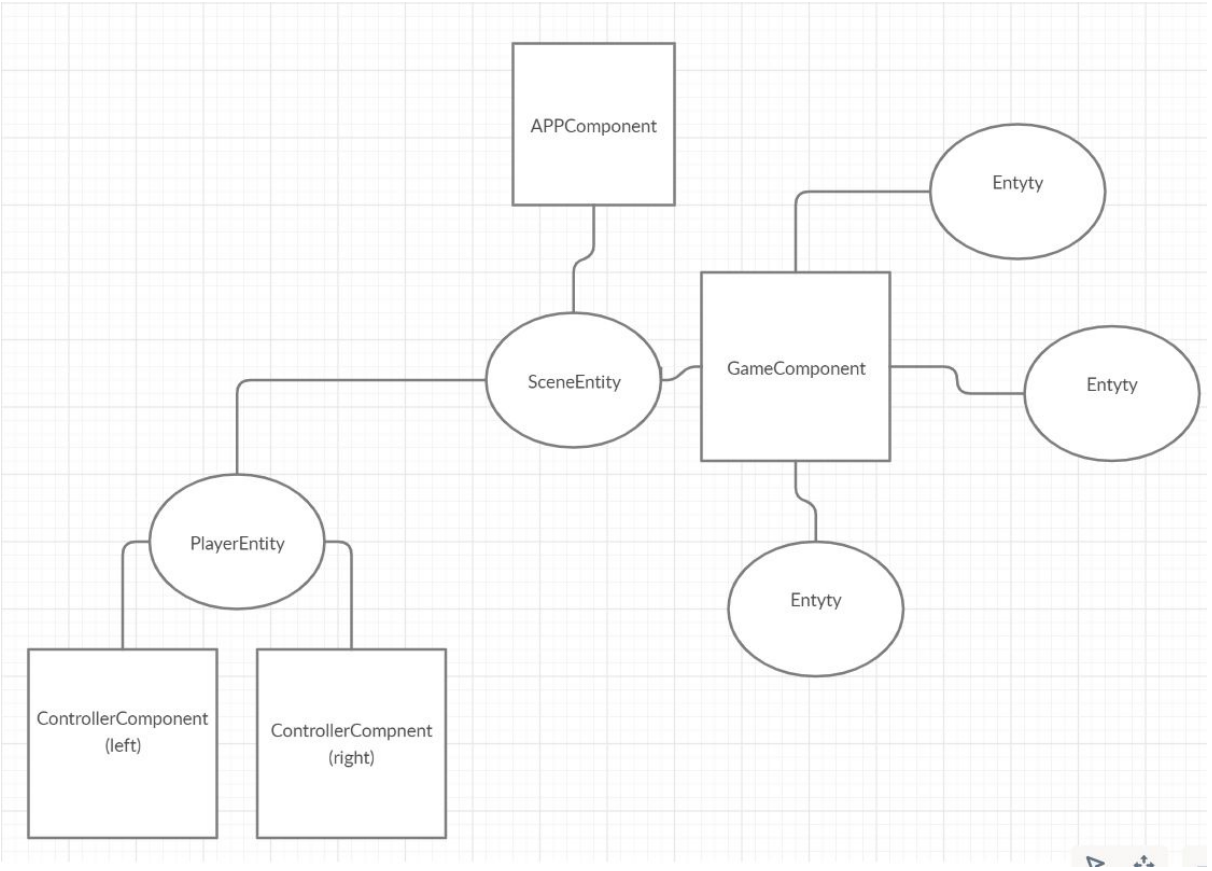


Ilustración 18 - Arquitectura angular

En esta imagen podemos ver representados por cubos los componentes de Angular y representados por óvalos las entidades de AFrame donde básicamente lo que hacemos es manejar el estado de dichas entidades en nuestros componentes pero además agrupamos distintas entidades en conjuntos lógicos como por ejemplo el componente del juego GameComponent agrupa dentro de si distintas entidades tales como todo el entorno del juego, el escenario, el comportamiento del escenario etc...

De esta forma se divide el juego en distintos ‘trozos’ o partes como pueden ser los distintos menús para seleccionar el modo de juego, el entorno del juego, el cubo que se tendrá que cortar o los controladores que tomarán forma de sable láser para que puedas cortar los cubos. Todos ellos son componentes de angular que a su vez están formados por más componentes de angular o bien por entidades de Aframe.

En segundo lugar están los servicios que a su vez se pueden dividir en dos tipos, los que obtienen datos de un API externo y los que encapsulan lógica de comunicación y comportamiento de ciertos componentes.

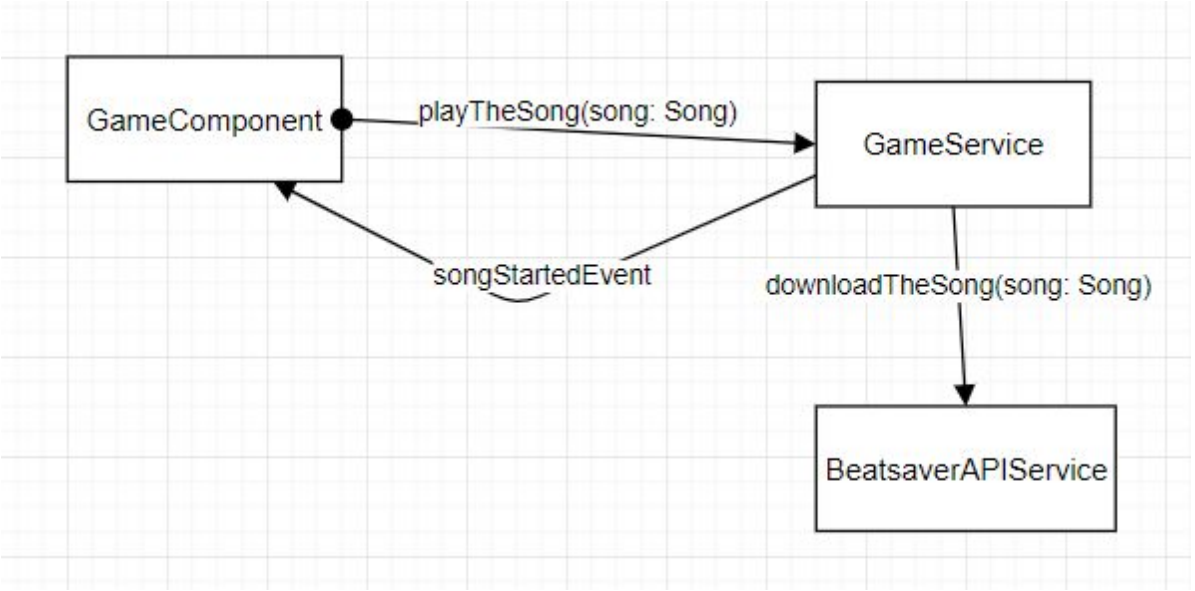


Ilustración 19 - Servicios en angular

Este podría ser un ejemplo simple como es la interacción entre el Componente y Servicio.

En tercer lugar tenemos los modelos, y esto es importante ya que trabajamos con TypeScript y la estructura de datos que nos ofrece el API no es necesariamente adecuado para lo que queremos hacer por lo que construimos nuestros propios modelos que describen bien la intención del código y los mapeamos con los datos que nos vienen del API.

Conclusiones

Resumiendo la arquitectura de nuestro videojuego se basa en utilizar las entidades de AFrame como elementos HTML dentro del patrón de desarrollo que nos proporciona el Framework de Angular. Esto nos proporciona algunas ventajas como que si ya conoces Angular el empezar a desarrollar experiencias VR en la web es algo muy fácil.

Funcionalidades

Como ya hemos visto en el apartado de las metodologías, nuestras funcionalidades las capturamos en Historias de usuario y en este apartado veremos más en detalle cada una de dichas historias de usuario, sin embargo,antes vamos a ver una visión global del juego para poder entender cada una de sus partes más pequeñas.

Resumen

Nada más entrar al juego nos encontramos con un panel donde podemos elegir distintos modos de juego, un solo jugador, competición con más gente, un tutorial y los créditos. En cada mano dispondremos de un puntero láser para poder seleccionar una de las opciones.

Si seleccionamos el modo de un solo jugador nos aparece un panel con una lista de canciones y tendremos que elegir una de dichas canciones, así como una de las dificultades disponibles para dicha canción y una vez que seleccionamos las dos cosas nos aparece la descripción de la canción y el nivel que vamos a jugar.

Pulsamos el botón ‘Play’ y desaparece el panel de selección del menú y ante nuestros ojos se construye el escenario por el cual van a venir hacia nosotros los cubos.

Una vez que se ha construido el escenario aparece el botón que dará comienzo a la canción y con ello empiezan a venir hacia nosotros los cubos que debemos cortar con los sables que tenemos en cada una de las manos.

Los cubos vienen señalados con la dirección con la que tenemos que cortarlos pero además cada uno de los cubos son de diferentes colores y cada cubo lo tendremos que cortar con el sable que sea del mismo color que el cubo.

Si cortamos un cubo de forma correcta se nos recompensa aumentando nuestra puntuación del marcador pero además si cortamos muchos cubos seguidos nuestra puntuación se multiplica.

Si fallamos muchos cubos en un intervalo corto de tiempo fallamos el nivel y tendremos que comenzar de nuevo o elegir otra canción.

Si llegamos al final de la canción sin fallar el nivel, se nos felicita por ello y podemos repetir la canción o elegir una nueva.

Historias de Usuario

En este apartado vamos a ver las principales historias de usuario que hemos seleccionado como el producto mínimo viable.

Ilustración 20 - Panel central del menú inicial

Tal y como podemos ver en esta Historia de Usuario la funcionalidad que nos descubre es la posibilidad de poder elegir distintos modos de juego y para ello implementamos un panel donde hay distintos botones que nos llevan a distintos modos de juego.

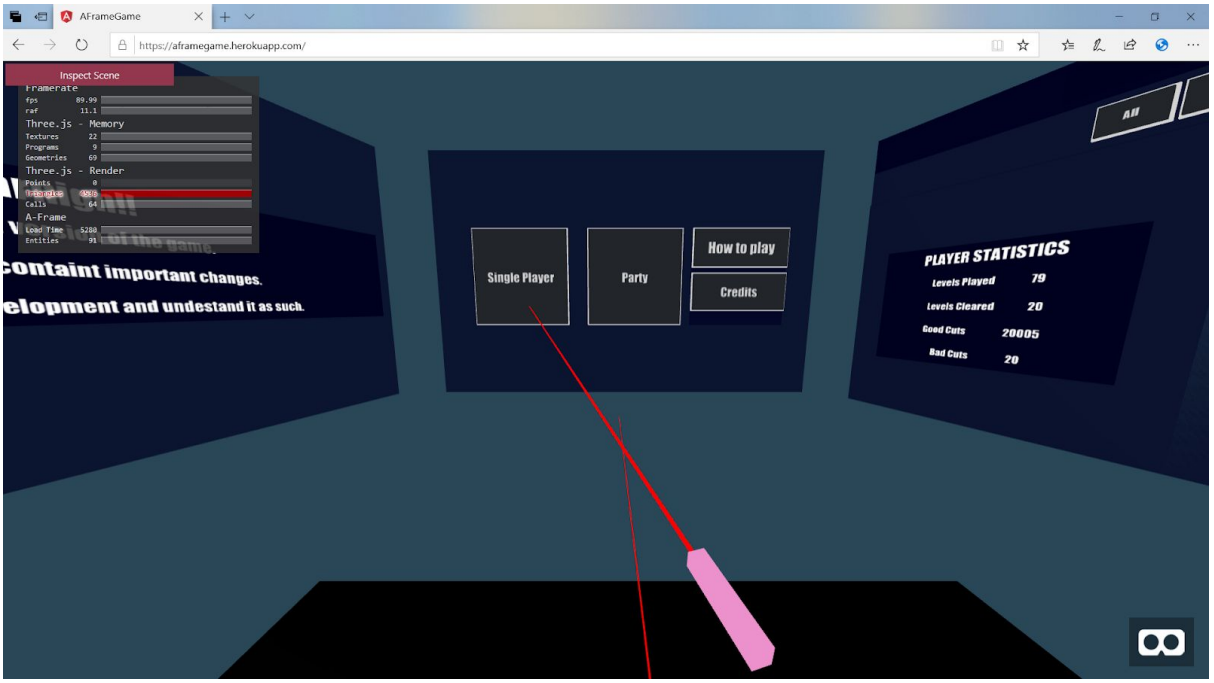


Ilustración 21 - Ejemplo panel central del menú inicial

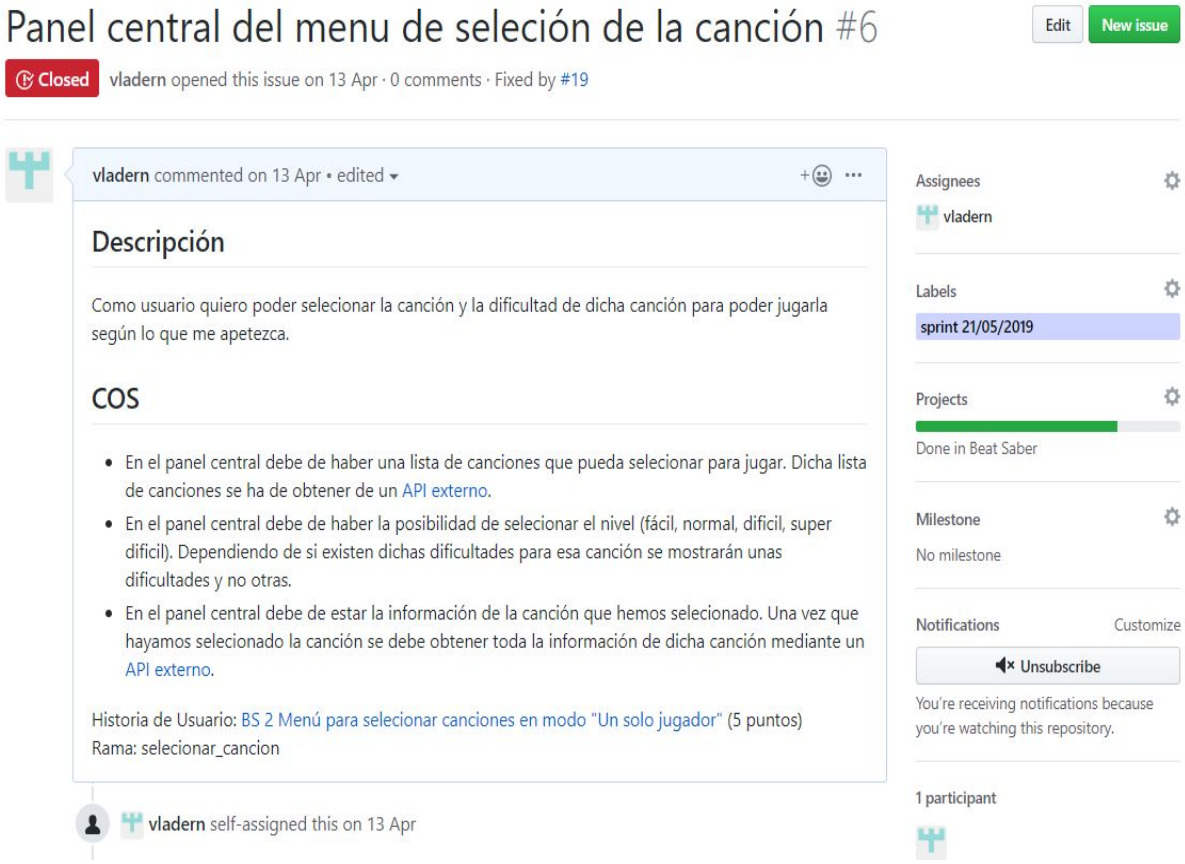


Ilustración 22 - Panel central del menú de selección

En la descripción de la historia de usuario podemos ver que se quiere implementar un menú para seleccionar una canción y una dificultad para dicha canción pero además en las condiciones de satisfacción podemos ver que esta lista de canciones ha de venir de un api externo y que además ha de haber información que describa la canción.

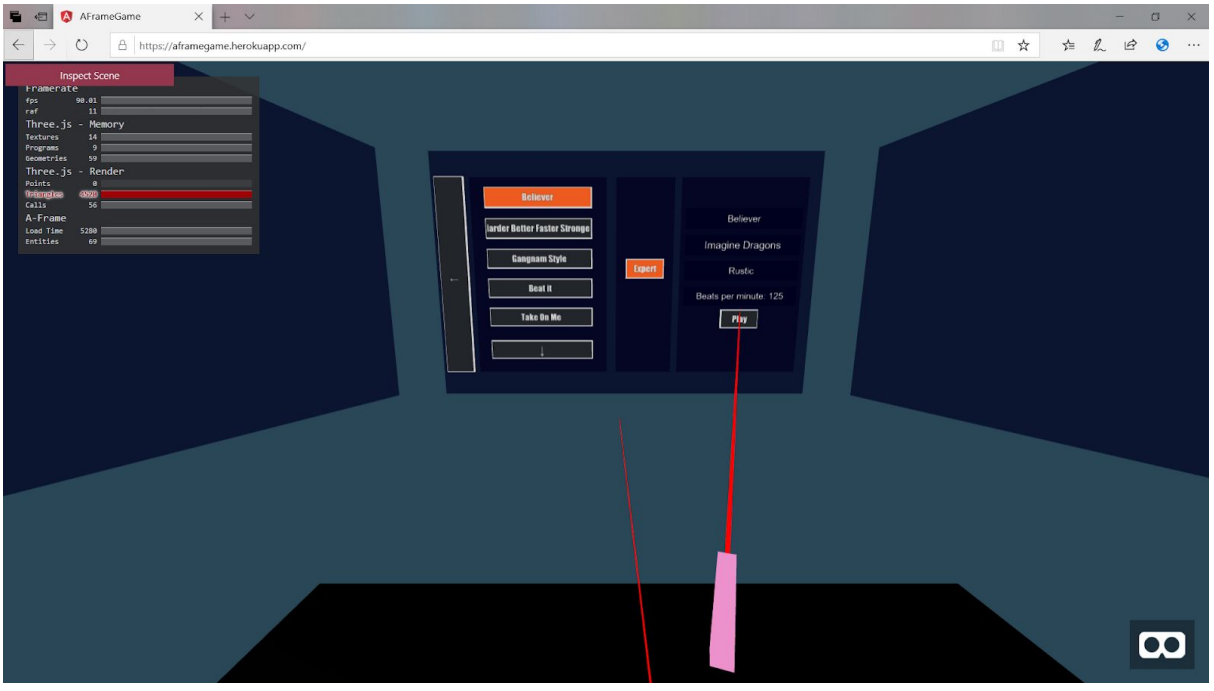


Ilustración 23 - Ejemplo panel central del menú de selección

Ilustración 24 - El jugador debe permanecer en el centro

En este caso lo que nos dice la historia de usuario es que el jugador ha de estar en el centro de una plataforma, por seguridad del propio jugador, y además en el caso de que el jugador se saliese de un área segura ha de aparecer un aviso que le diga al jugador que vuelva al centro.

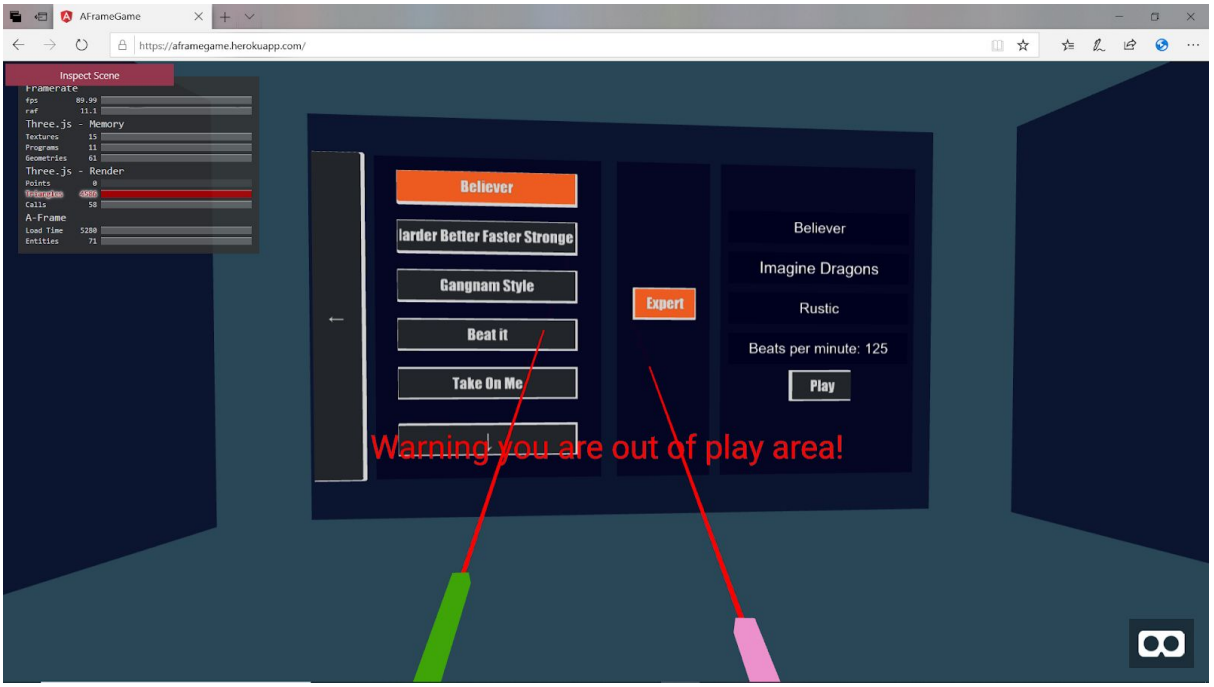


Ilustración 25 - Ejemplo jugador debe permanecer en el centro

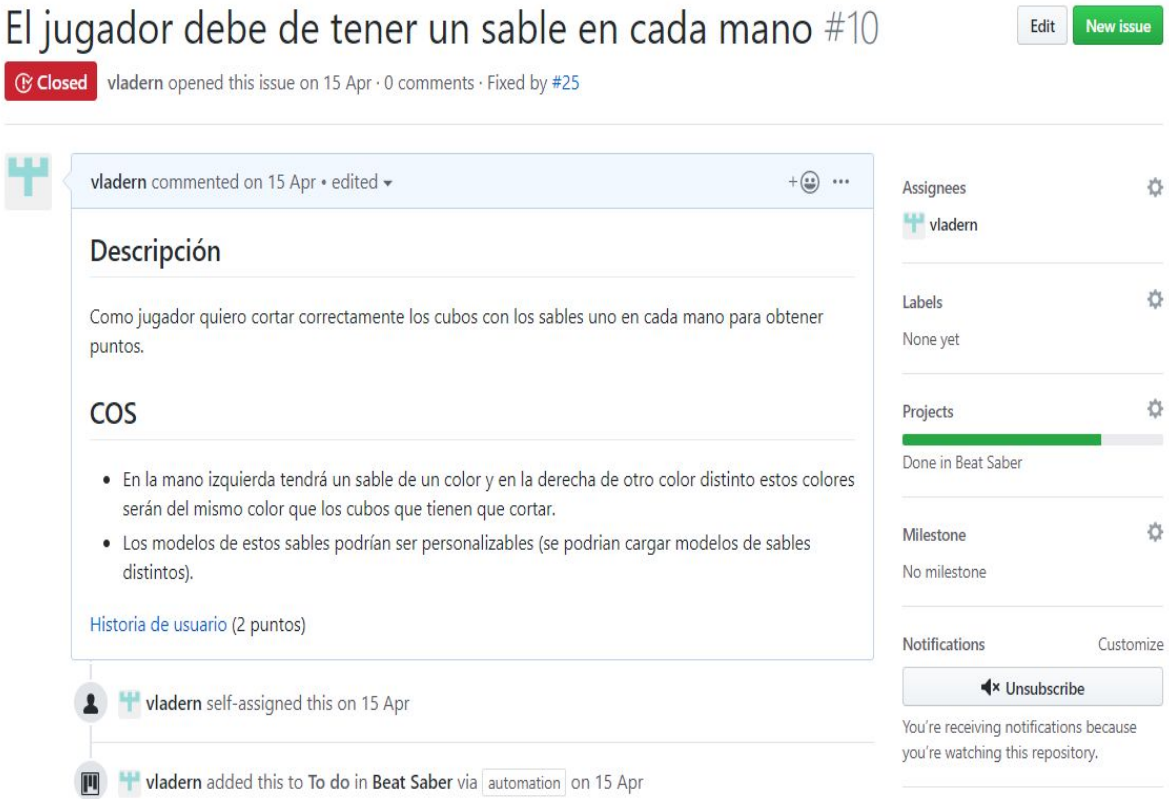


Ilustración 26 - El jugador debe de tener un sable en cada mano

Para la siguiente historia lo que se quiere es que el jugador cuando va a jugar tenga dos sables en cada mano, estos sables han de ser de distintos colores y que estos colores han de ser los mismos que los de los cubos que va a cortar.

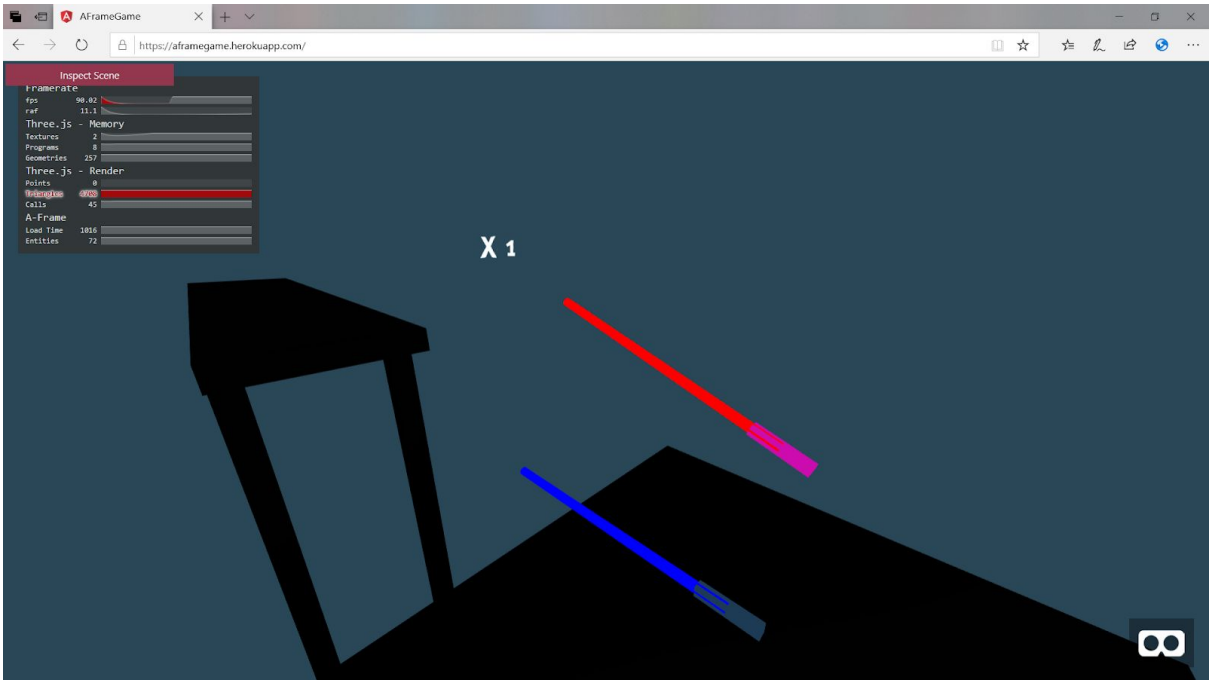


Ilustración 27 - Ejemplo, el jugador debe tener un sable en cada mano

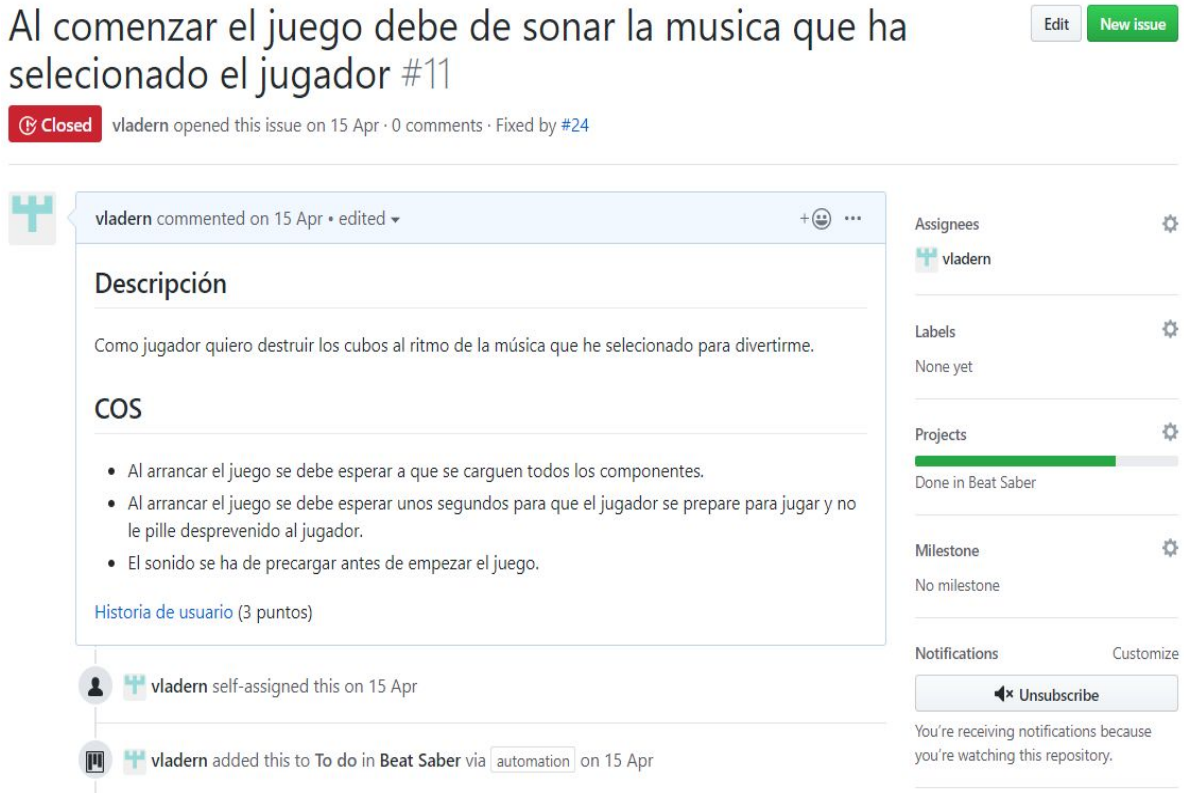


Ilustración 28 - Debe de sonar la música que ha seleccionado el jugador

Básicamente lo que podemos ver en esta historia es que para que el juego sea divertido la música ha de sonar y nosotros cortaremos los cubos al ritmo de la misma. Pero el juego no ha de empezar repentinamente sino que se ha de dar un poco de tiempo al jugador para situarse.

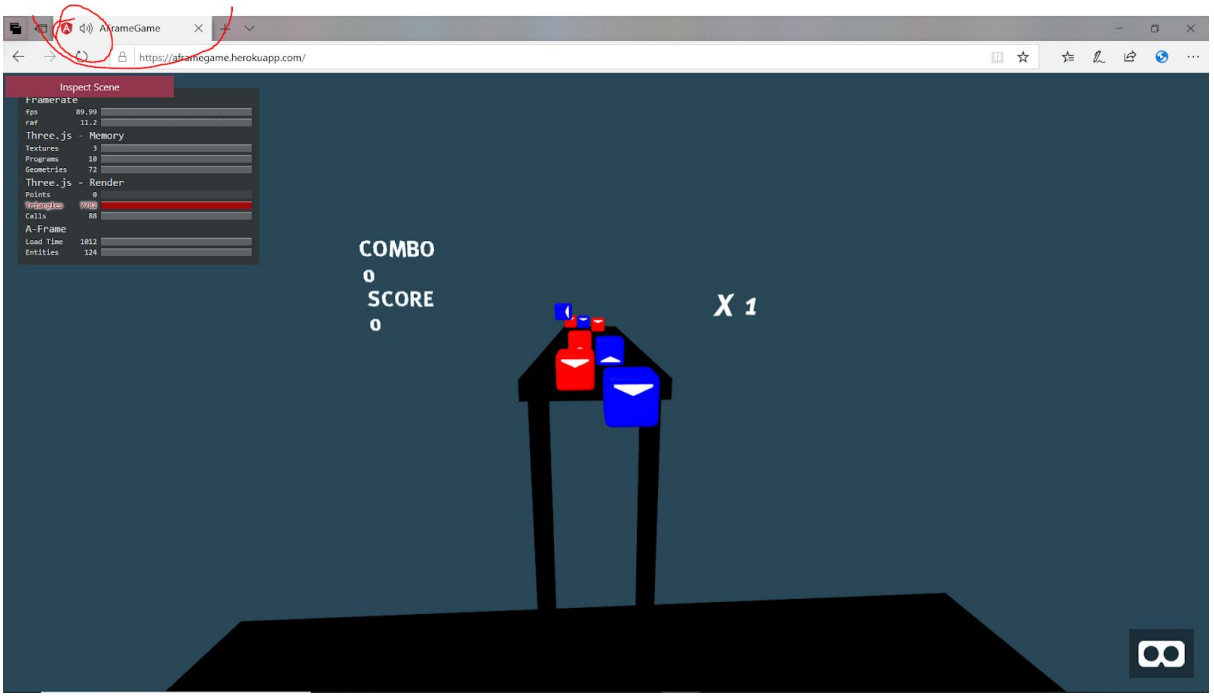


Ilustración 29 - Ejemplo debe sonar música que ha seleccionado el jugador

Ilustración 30 - El jugador debe interactuar con los cubos

Esta es quizá la funcionalidad más importante del juego, ya que el jugador ha de interactuar de alguna forma con los cubos, y esto ha de suceder de la forma en la que se describe en las condiciones de satisfacción. Primero los cubos han de venir hacia el jugador, y posteriormente cada cubo ha de tener indicada la dirección en la que se tiene que cortar y por último es que se ha de poder detectar en qué dirección se ha cortado el cubo.

Ilustración 31 - Ejemplo el jugador debe interactuar con los cubos

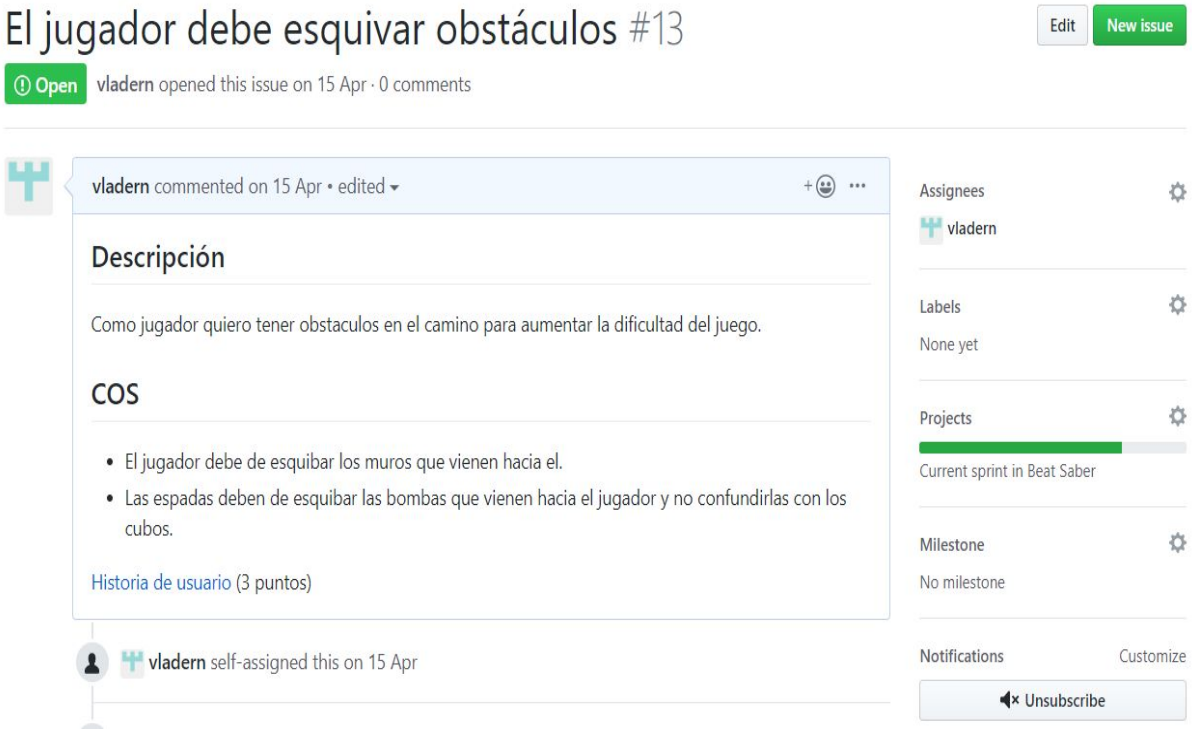


Ilustración 32 - El jugador debe esquivar obstáculos

La siguiente funcionalidad trata de aumentar la dificultad del juego poniendo obstáculos como muchos que vienen hacia él para que el jugador tenga que esquivarlos. Funcionalidad que por desgracia no me ha dado tiempo a implementar.

Ilustración 33 - Recibir puntos

En esta funcionalidad lo que se quiere es recibir puntuación por los cubos que cortamos bien. Pero no solo eso, sino que también se quiere que si tienes una racha y cortas muchos cubos seguidos esa puntuación se multiplique por lo que resulte ser mucho más rentable cortar muchos cubos seguidos, pero si cortas mal un cubo esta racha se acaba y toda esta información se ha de mostrar al jugador.

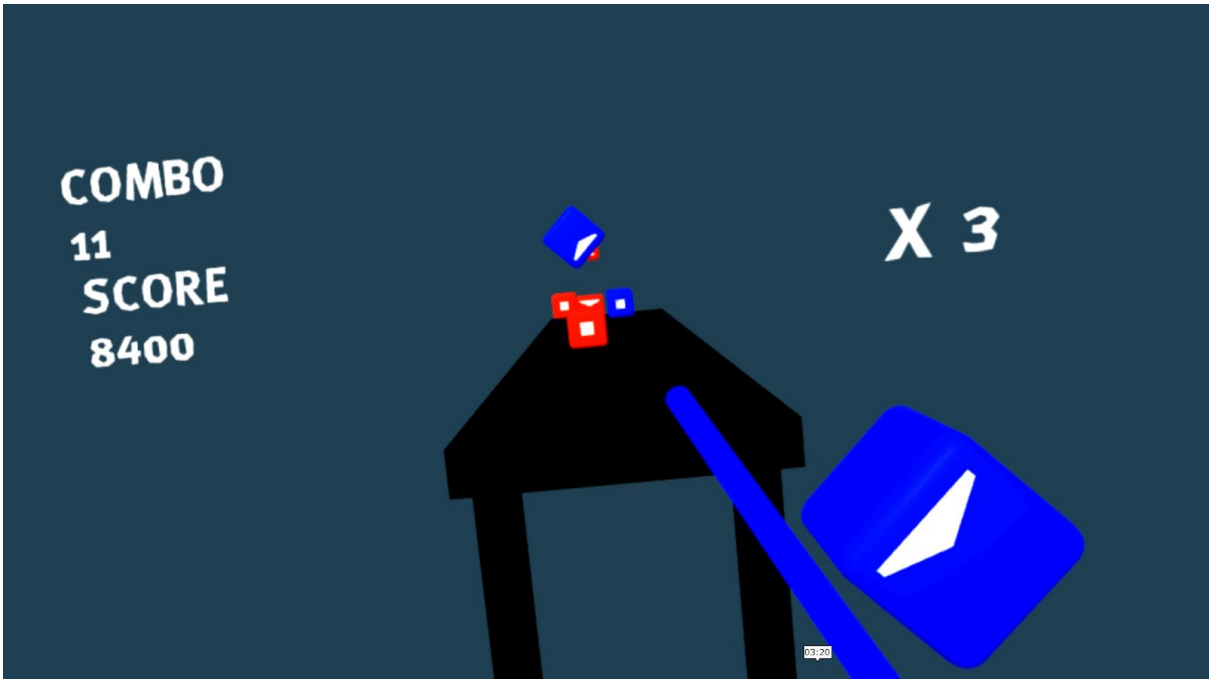


Ilustración 34 - Ejemplo recibir puntos

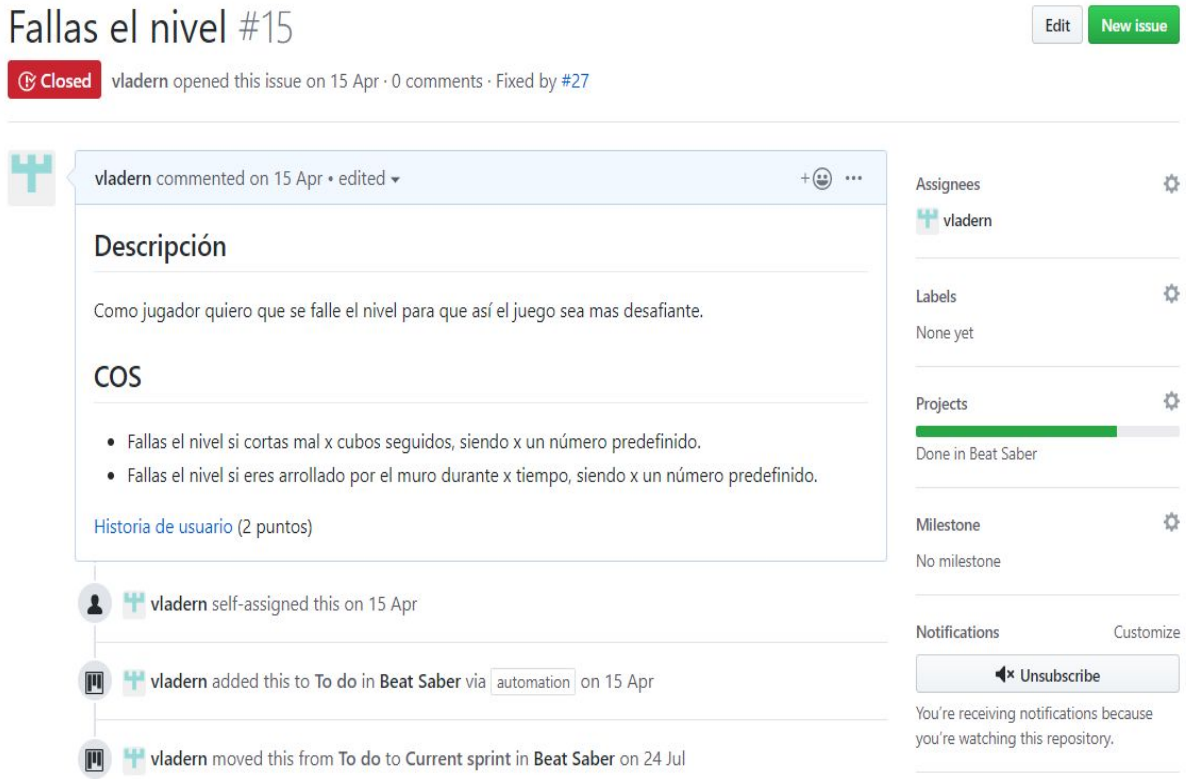


Ilustración 35 - Fallas el nivel

En esta ocasión lo que se pretende es que si cortas mal o no cortas muchos cubos seguidos o más bien en un intervalo de tiempo entonces que el nivel se de por fallido y que lo tengas que repetir para que el juego tenga que ser más desafiante.

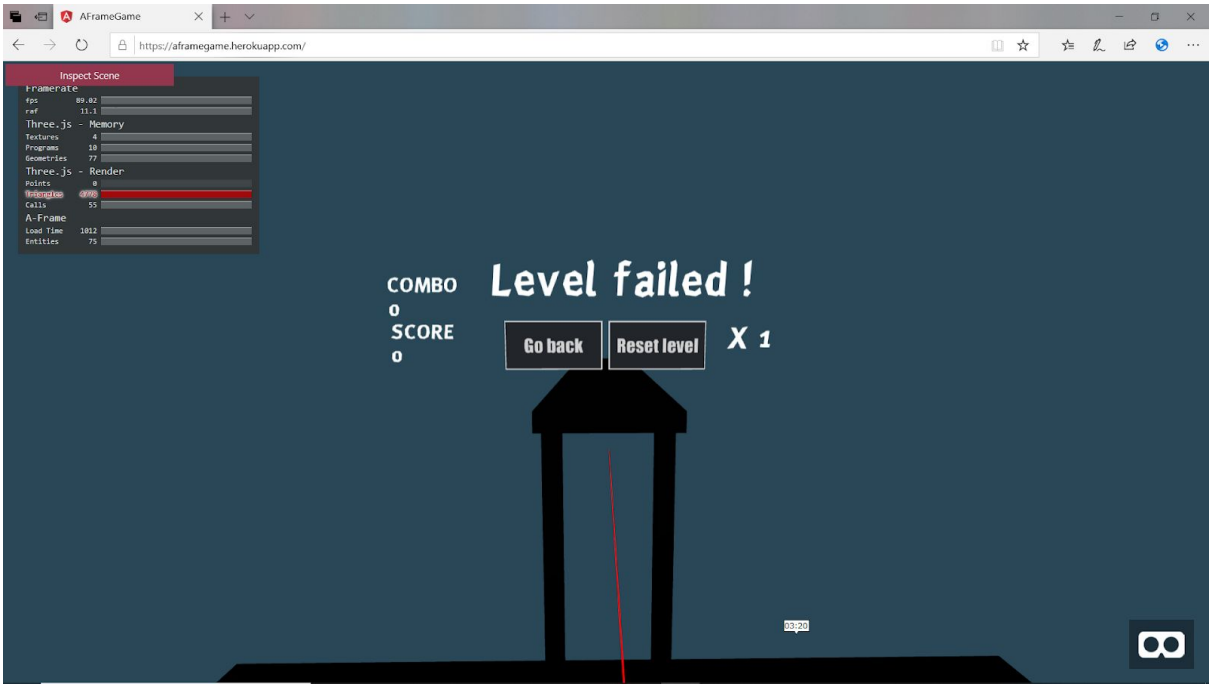


Ilustración 36 - Ejemplo fallas el nivel

Testing: pruebas

Dado que desde el principio del proyecto se ha intentado llevar a cabo TDD (Desarrollo Dirigido por Tests) antes de nada tendremos que ver más en profundidad qué significa trabajar de esta forma.

Para empezar vamos a ver cual es el enfoque tradicional y cual es el enfoque del TDD respecto al testing.

En el enfoque tradicional los test sirven para comprobar que no hay errores en el código, en el enfoque TDD los test se utilizan para ayudar a hacer la especificación y el desarrollo. En el enfoque tradicional los test prueban que el código hace lo que se ha especificado, en el TDD son ejemplos de cómo ha de funcionar el código. En el enfoque tradicional primero se especifica, luego se programa y por último se prueba, en TDD se diseñan los test al mismo tiempo que la especificación y después se programa.

Existen diferentes tipos de test, como por ejemplo los tests unitarios que prueban una pequeña parte del programa, el test de integración que prueba código, el cual depende de recursos externos (APIs, bases de datos, emails etc...), los test funcionales que simulan a un usuario final del programa y por último las pruebas de estrés que prueban el rendimiento del sistema.

En este proyecto hemos realizado solo test unitarios y test de integración ya que el resto de test eran muy complicados para aplicarlos a un videojuego y que siguieran siendo eficaces y efectivos.

Test Unitarios

Existen distintos enfoques de lo que debería ser un test unitario. Una visión estricta y la otra visión de XP y TDD. En la visión estricta una unidad a probar es una clase aislada lo cual significa que se sustituyen las relaciones con otras clases, con mocks y stubs de forma que si falla un test sabemos con seguridad que la responsable de que falle es la propia clase. Esta visión está muy orientada al control de calidad del código.

Por otro lado nos encontramos con la visión de XP y TDD donde un test unitario invoca una funcionalidad concreta y realiza una única comprobación sobre la conducta del sistema.

Este tipo de tests se suelen llamar *end-to-end .* En este caso no se hace tanto énfasis en que el test esté aislado,ya que nuestro test puede llamar a otras unidades que ya habrán sido probadas porque han sido desarrolladas con TDD. El test debe de ser repetible e independiente y ha de probar una pequeña funcionalidad del programa. Ha de poder ejecutarse en local, no ha de provocar efectos colaterales en el resto de test y lo más importante el test ha de ser sencillo, legible y mantenible.

Nombres correctos para los test

Al igual que con los tests unitarios hay dos formas de darles nombre a los test.

La primera forma de dar nombre se suele utilizar en la visión tradicional. Tal y como se puede observar se pone una preposición en este caso C1A (Caso de prueba uno A) y luego el nombre de la función a la que se va a probar.

```
16      @Tag("TestsLlanosTablaA")
17      @Test
18      void C1A_buscarTramoLlano() {
19          this.listaDeLecturas.add(3);
20          this.resultadoEsperado = new Tramo(0,0);
21
22          this.resultadoObtenido =  this.llanos.buscarTramoLlanoMasLargo(this.listaDeLecturas);
23          assertAll("C1A_buscarTramoLlano",
24                  ()-> assertEquals(this.resultadoEsperado.getOrigen(), this.resultadoObtenido.getOrigen()),
25                  ()-> assertEquals(this.resultadoEsperado.getLongitud(), this.resultadoObtenido.getLongitud()
26          );
27
28          assertEquals(this.resultadoEsperado, this.resultadoObtenido);
29      }
```

Ilustración 37 - Ejemplo nombre test tradicional

Y la segunda forma es la que se suele utilizar en un testing orientado a la funcionalidad. Donde el nombre del test no es un nombre como tal sino más bien una descripción literal de la funcionalidad que se está probando.

```
43  it('should create', () => {
44    expect(component).toBeTruthy();
45  });      You, 5 months ago • escenario simple del menu
46
47  it('On init the component shold get the songs', fakeAsync(() => {
48    expect(component.songsToBeShown.length).toBeGreaterThan(0);
49  }));
50
51  it('When a song is selected, the panel with dificulties should apear', fakeAsync(() => {
52    const selectedSong: Song = component.songsToBeShown[0];
53    component.selectTheSong(selectedSong);
54    fixture.detectChanges();
55    const dificultiesPanel = fixture.debugElement.nativeElement.querySelector('#levelSelectionPanel');
56    expect(dificultiesPanel).toBeTruthy();
57  }));
```

Ilustración 38 - Ejemplo nombre test orientado a la funcionalidad

Fases del TDD

Se puede decir que el TDD comienza cuando capturamos la funcionalidad de nuestro proyecto, en nuestro caso lo hemos hecho con la historia de usuario donde hemos especificado las COS (Condiciones De Satisfacción).

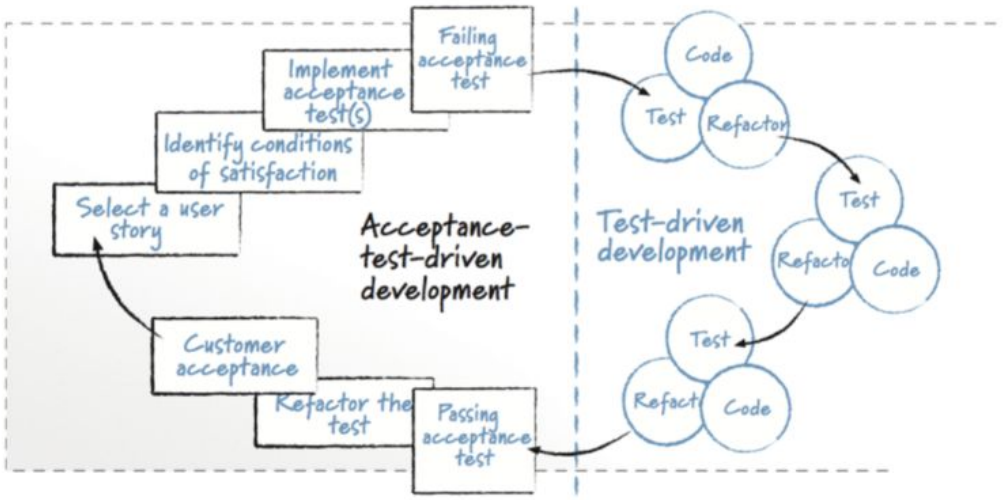


Ilustración 39 - Ejemplo tdd

Y así es como comenzamos escribiendo un test que comprueba la funcionalidad que queremos añadir a nuestro sistema, dicho test ha de fallar ya que la funcionalidad no ha sido implementada todavía.

Después implementamos la funcionalidad **mínima** que haga que el test pase, a pesar de que el código no sea el más limpio posible.

Luego refactorizar el código.

Ahora escribimos un nuevo test, sobre la misma funcionalidad del primero pero tenemos que hacer que dicho test falle (alguna condición o caso de uso que no haya sido implementado con el primer test).

Hacemos que el test pase y refactorizamos. Y continuamos de esta forma hasta que satisfacemos el COS por completo.

Jasmine y Karma

Una vez que hemos visto que es el testing, como encaja TDD dentro de este y cuales han sido los pasos que hemos hecho para desarrollar el proyecto vamos a hablar concretamente sobre la herramienta que utiliza Angular por defecto para realizar *Testing* .

Jasmine

Jasmine es un *open source behaviour driven development* (también soporta el TDD) framework para hacer testing en Javascript. Presume de correr en todas las plataformas de desarrollo para Javascript y de tener una sintaxis clara y fácil de leer. Algunas de sus caracteristicas son que soporta testing asíncrono y hace uso de *'spies'* para la implementación de dobles. Es independiente del navegador, framework o plataforma.

Bien, dado que este framework presume de tener una sintaxis clara vamos a ver un poco de código. Si miramos dentro de la carpeta src/app/components vamos a encontrar **app.component.spec.ts**

Ilustración 40 - Ejemplo Jasmine

Vamos a ir paso por paso para poder entenderlo todo:
Importamos TestBed, Async y nuestro componente para poder ejecutar los test, también CUSTOM_ELEMENTS_SCHEMA y NO_ERROR _SCHEMA para eliminar ciertas incompatibilidades a la hora de usar Aframe.

TestBed es una de las *utilities* más importantes para poder hacer *testing* en Angular. Crea un módulo Angular que se configura con configureTestingModule (tal y como se puede ver en la línea 9) para producir el entorno del módulo Angular para la clase que se va a testear.

En otras palabras lo que hace es separar el componente que vamos a probar de su propio módulo y lo conecta a un módulo generado dinámicamente y adaptado específicamente para estas pruebas.

En la línea 17 podemos ver una función “beforeEach”, esta función se va a ejecutar cada vez que se lanze un nuevo test y dentro de esta función podemos ver como creamos el componente a testear y lo instanciamos.

En la línea 23 vemos la función “it” en su primer parámetro, que recibe el texto e imprime el test al ejecutarse y luego recibe una función que genera un “**async**” con el contenido que testea nuestra funcionalidad. Este primer test lo que prueba es que el componente se ha podido crear e instanciar bien con “**toBeTruthy**” .

En la línea 28 lo que probamos es que un elemento html exista dentro del DOM de nuestro componente. Primero buscamos dicho elemento por su id con “**querySelector”** y luego con “**toBeTruthy**” comprobamos que existe.

Karma

Karma es un *test runner open source* , desarrollado y mantenido por el equipo de Angular que nos permite automatizar algunas tareas de frameworks de test como Jasmine. El objetivo principal de Karma es brindar un entorno productivo para los desarrolladores. Uno donde no has de complicarte con muchísimas configuraciones sino que es uno donde escribes código y obtienes *feedback* instantáneo de tus pruebas.

Para empezar a utilizarlo lo único que tenemos que hacer es escribir “ng test” en la consola dentro de la carpeta de nuestro proyecto, se abre el navegador configurado (por defecto es Chrome pero este se puede cambiar por cualquier navegador del mercado) y vemos cómo se ejecutan nuestras pruebas en dicho navegador.

```
Edge 18.17763.0 (Windows 10.0.0): Executed 3 of 58 SUCCESS (0 secs / 0.786 secs)
LOG: '%ccomponents:renderer:warn %cComponent `antialias` is deprecated. Use `renderer="antialias: true"` instead.%c
LOG: 'THREE.WebGLRenderer', '103dev'
Edge 18.17763.0 (Windows 10.0.0): Executed 4 of 58 SUCCESS (0 secs / 0.97 secs)
LOG: '%ccomponents:renderer:warn %cComponent `antialias` is deprecated. Use `renderer="antialias: true"` instead.%c
or: orange'
Edge 18.17763.0 (Windows 10.0.0): Executed 4 of 58 SUCCESS (0 secs / 0.97 secs)
LOG: '%ccomponents:renderer:warn %cComponent `antialias` is deprecated. Use `renderer="antialias: true"` instead.%c
LOG: 'THREE.WebGLRenderer', '103dev'
Edge 18.17763.0 (Windows 10.0.0): Executed 5 of 58 SUCCESS (0 secs / 1.157 secs)
LOG: '%ccomponents:renderer:warn %cComponent `antialias` is deprecated. Use `renderer="antialias: true"` instead.%c
or: orange'
Edge 18.17763.0 (Windows 10.0.0): Executed 5 of 58 SUCCESS (0 secs / 1.157 secs)
LOG: '%ccomponents:renderer:warn %cComponent `antialias` is deprecated. Use `renderer="antialias: true"` instead.%c
LOG: 'THREE.WebGLRenderer', '103dev'
Edge 18.17763.0 (Windows 10.0.0): Executed 6 of 58 SUCCESS (0 secs / 1.334 secs)
LOG: '%ccomponents:renderer:warn %cComponent `antialias` is deprecated. Use `renderer="antialias: true"` instead.%c
or: orange'
Edge 18.17763.0 (Windows 10.0.0): Executed 6 of 58 SUCCESS (0 secs / 1.334 secs)
LOG: '%ccomponents:renderer:warn %cComponent `antialias` is deprecated. Use `renderer="antialias: true"` instead.%c
Edge 18.17763.0 (Windows 10.0.0): Executed 7 of 58 (skipped 51) SUCCESS (1.603 secs / 1.549 secs)
TOTAL: 7 SUCCESS
TOTAL: 7 SUCCESS
```

Ilustración 41 - Ejecución test con Karma

En este caso podemos ver que las 7 pruebas que tenemos dentro del “**AppComponent**” pasan satisfactoriamente.

Muy simple verdad, bueno vamos a entender un poco mejor qué es lo que está pasando. Y es que nuestro proyecto Angular por defecto tiene configurado que los archivos ****.spec.ts** contendrán test y lo que hace carma es buscar y lanzar dichos test.

Instalación del proyecto

Antes que nada tenemos que descargarnos el proyecto del repositorio remoto de git con la siguiente url: https://github.com/vladern/AFrameGame. Lo podemos hacer desde el propio Github o ejecutando el comando ***git clone https://github.com/vladern/AFrameGame.git .***

Lo siguiente es descargar e instalar Node.js en nuestro ordenador, si no lo tenemos ya. Esto es tan simple como seguir la instrucciones de su página web oficial https://nodejs.org/es/ .

Una vez que tenemos descargado e instalado el node lo que vamos a hacer es ejecutar el siguiente comando: ***npm install -g @angular/cli*** y con esto lo que hacemos es instalar el angular cli en su última versión. Por último, ejecutamos en la terminal ***ng serve*** y tendremos un servidor web escuchando en ***localhost:4200*** . Si todo esto es demasiado complicado para nosotros lo que podemos hacer es entrar al siguiente link : https://aframegame.herokuapp.com/ en dicha url se encuentra el proyecto desplegado en su última versión.

Finalmente lo único que queda es entrar al navegador y disfrutar del juego, mi recomendación es que lo hagamos o bien con **Firefox** o bien con **Edge** ya que los navegadores basados en **Chromium** no tienen habilitado el WebVr por defecto y no siempre funciona del todo bien.

Conclusiones

A modo de conclusión en este apartado vamos a repasar los objetivos planteados al principio del proyecto y ver si los hemos logrado alcanzar, qué dificultades hemos tenido y cuáles podrían ser las posibles mejoras para la solución que hemos dado.

Integración de frameworks

El objetivo de integrar el framework de AFrame con el frameworks de Angular, ha sido todo un éxito. Ya hemos podido trabajar libremente con estos dos frameworks casi sin ningún tipo de problema. Se han podido implementar funcionalidades prácticamente del mismo modo que implementamos una aplicación web, pero con la potencia que nos ofrece AFrame para crear elementos 3D.

Las únicas dificultades que he tenido han sido las de configuración, como por ejemplo que cada vez que instalas una nueva librería para un nuevo componente de AFrame esta se tenía que importar manualmente en el fichero de "**polifils.ts**"

```
54  /***************************************************************************************************
55   * APPLICATION IMPORTS
56   */
57  import 'aframe';
58  import '../init_scripts/register-ng-primitives';
59  import 'aframe-extras';
60  import 'aframe-controller-cursor-component';
61  import 'aframe-physics-system';
62  import 'aframe-aabb-collider-component';
63  import 'aframe-html-shader';
64  import 'aframe-gui';
65  /***************************************************************************************************
```

del proyecto para que este fuera detectado en los componentes de Angular y lo mismo para AFrame, cada vez que creaba un nuevo componente de Angular este se tenía que registrar en el framework de AFrame.

```
9   AFRAME.registerPrimitive('a-game', { mappings: {} }); //<- specify all your ng tags starting here
10  AFRAME.registerPrimitive('a-menu', { mappings: {} });
11  AFRAME.registerPrimitive('a-single-player-menu', { mappings: {} });
12  AFRAME.registerPrimitive('a-party-menu', { mappings: {} });
13  AFRAME.registerPrimitive('a-how-to-play', { mappings: {} });
14  AFRAME.registerPrimitive('a-credits', { mappings: {} });
15  AFRAME.registerPrimitive('a-beat', { mappings: {} });
16  AFRAME.registerPrimitive('a-controller', { mappings: {} });
```

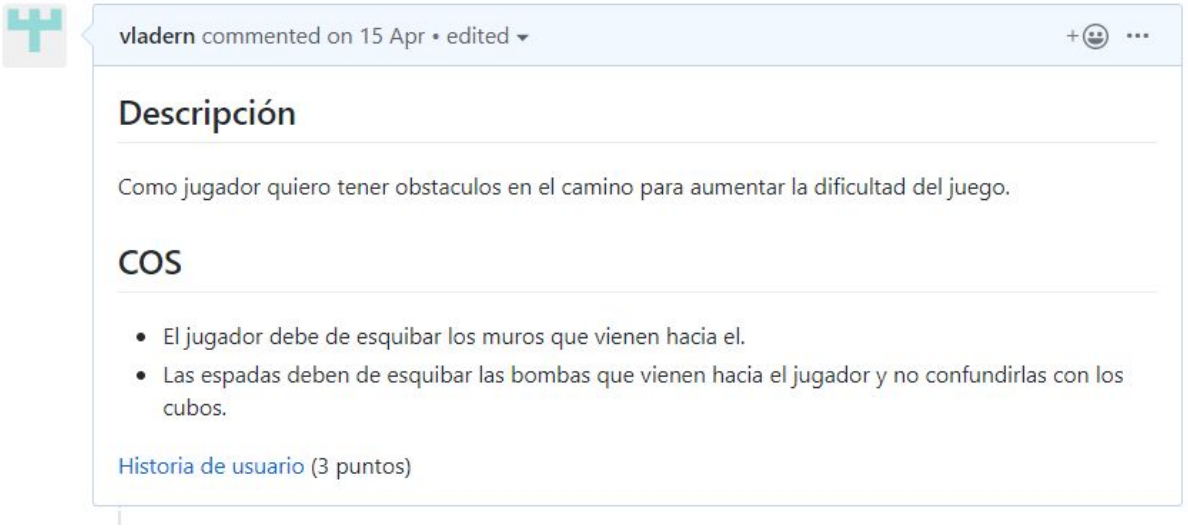
Y claro cada vez que creas un nuevo componente Angular o quieres utilizar alguna entidad o componente AFrame que has visto en NPM[41] es necesario actualizar estos archivos manualmente.

Quizá una posible solución a futuro es ampliar los comandos para instalar librerias de Aframe y crear componentes angular para que estos archivos se actualicen manualmente.

[41] NPM, sistema de gestión de paquetes para node, sitio oficial

Idea general y mecánicas del videojuego

Dado que ya existen juegos muy parecidos al nuestro tales como Beat Saber[42],lo cual conlleva a darnos una idea general así como de las mecánicas del juego. Pero aún quedaba capturar dichos requisitos y esto lo hemos hecho en forma de historias de usuario.



Esta forma de especificar la funcionalidad del juego nos facilita mucho su implementación ya que sabemos lo que queremos implementar y cuales son las condiciones para que se dé por hecha dicha funcionalidad.

Creación de tareas

La creación de tareas nos la ha facilitado el framework de Scrum que especifica explícitamente cómo han de ser especificadas. Además GitHub tiene las herramientas específicas para su implementación tales como los "*Projects*"[43] y los "*Issues*"[44]. Dentro de Projects puedes crear proyectos y dentro de uno de estos proyectos se pueden crear columnas para visualizar el flujo de trabajo de un proyecto.

Para crear una tarea lo único que tenemos que hacer es un nuevo *Issue* e incluirlo dentro del proyecto.

Implementación de las tareas

Para la implementación de las tareas hemos utilizado el TDD que es una práctica del XP de los cuales ya hemos hablado extendidamente en los apartados anteriores. Realmente me ha gustado mucho la experiencia de desarrollar de esta forma,ya que te proporciona una seguridad increíble a la hora de incorporar nuevas funcionalidades al proyecto, y con una simple ejecución de todos los test sabes que nada se ha roto y que todo sigue funcionando como debe.

[42] Beat Saber, juego de realidad virtual, sitio oficial
[43] *Projects,* tablero kanban de GitHub, sitio oficial
[44] *Issues,* tarea de un kanban de GitHub, sitio oficial

Otro aspecto son las funcionalidades que se han podido implementar, como ya hemos visto al principio del proyecto definimos un producto mínimo viable (MVP[45]) que estaba compuesto por una serie de funcionalidades y me enorgullezco de poder decir que se han implementado todas las funcionalidades de dicha lista.

Resumen

En resumidas cuentas, se han alcanzado todos los objetivos propuestos al principio del proyecto, con algunos contratiempos como bugs que no se han podido solucionar (todavía), o cambios en la interfaz del API externo, que llevó un tiempo de investigación de cuáles eran los nuevos *endpoints*[46].

[45] MVP, *minimum viable product*
[46] *Endpoints*, rutas o enlaces con sus parámetros para hacer una petición http.

Referencias y bibliografía

- Proyectos
  - Cliente web: https://github.com/vladern/AFrameGame
- Tecnologías y herramientas utilizadas
  - Trello: https://trello.com
  - GitHub: https://github.com
  - Angular: https://angular.io
  - AFrame: https://aframe.io
  - Typescript: https://www.typescriptlang.org
  - Javascript: https://www.javascript.com
  - Node.js: https://nodejs.org/es/
  - Visual Studio Code: https://code.visualstudio.com
  - Slack: https://slack.com
  - Platzi: https://platzi.com
- Material de apoyo
  - Documentación de AFrame https://aframe.io/docs/0.9.0/introduction/
  - Documentación de Angular https://angular.io/docs
  - Curso sobre Realidad Virtual en la Web de https://platzi.com (por desgracia han descontiuado el curso y ya no se encuentra disponible)
  - Aframe workshop https://github.com/fcor/aframe-workshop
  - Integración de Aframe y Angular https://github.com/shane-lab/ng2aframe
  - Documentación ágil https://www.javiergarzas.com/2010/03/documentacion-agil.html
  - Introducción al *testing* con angular https://medium.com/@jorgeucano/introducción-al-testing-en-angular-da415ef8c47
  - Angular 7 Unit Testing https://medium.com/@wiem.khelifii/angular-7-unit-testing-lazy-loading-step-by-step-6c2c20436f5c
  - TDD pruebas ágiles https://github.com/domingogallardo/apuntes-mads/blob/master/sesiones/06-tdd-pruebas-agiles/tdd-pruebas-agiles.md
  - Manifiesto ágil https://github.com/domingogallardo/apuntes-mads/blob/master/sesiones/03-manifiesto-agil/manifiesto-agil.md
  - Historias de usuario https://github.com/domingogallardo/apuntes-mads/blob/master/sesiones/09-historias-de-usuario/historias-de-usuario.md
  - Kanban y Scrum https://github.com/domingogallardo/apuntes-mads/blob/master/sesiones/13-kanban-y-scrum/kanban-y-scrum.md